

# IRIS (Integrity and Reliability in Integrated Circuits) Test Article Generation (ITAG)

Final Report

March 31, 2015

Period of Performance: February 14, 2011 – February 28, 2015

Contract Number: HR0011-11-C-0041

Principal Investigator: Dr. Jeffrey Draper

Co-Principal Investigator: Mr. Matthew French

## Submitted to:

Defense Advanced Research Projects Agency

Contracts Management Office

675 North Randolph Street, Arlington, VA 22203-2114

Attention: Jessica Downing; Tel: (571) 218-4961; [jessica.downing@darpa.mil](mailto:jessica.downing@darpa.mil)

Kerry Bernstein; Tel: (703) 526-2117; E-mail [Kerry.Bernstein@darpa.mil](mailto:Kerry.Bernstein@darpa.mil)

## Submitted by:

University of Southern California – Information Sciences Institute

4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292

Technical Points of Contact	Administrative Point of Contact
Dr. Jeffrey Draper, <a href="mailto:draper@isi.edu">draper@isi.edu</a> Mr. Matthew French, <a href="mailto:mfrench@isi.edu">mfrench@isi.edu</a>	Ms. Brigidann Cooper, <a href="mailto:brigidannc@usc.edu">brigidannc@usc.edu</a>
University of Southern California Information Sciences Institute 4676 Admiralty Way, Suite 1001 Marina del Rey, CA 90292 Tel: 310-448-8750, FAX: 310-823-6714	University of Southern California 3720 S. Flower Street Los Angeles, CA 90089-0701 Tel: (310) 448-9161, FAX: (213) 740-6070
Lead Organization and Reference Numbers: DUNS: 072933393; CAGE: 06SU2	University of Southern California, Information Sciences Institute
Type of Business of Lead Organization	Other Educational

Sponsored by

Defense Advanced Research Projects Agency

Microsystems Technology Office (MTO)

Issued by DARPA/CMO under Contract No. HR0011-11-C-0041

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. government.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to the Department of Defense, Executive Service Directorate (0704-0188). Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p><b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.</b></p>						
1. REPORT DATE (DD-MM-YYYY) 21/09/2015		2. REPORT TYPE FINAL		3. DATES COVERED (From - To) 14/02/2011 - 28/02/2015		
4. TITLE AND SUBTITLE IRIS (Integrity and Reliability in Integrated Circuits) Test Article Generation (ITAG)				5a. CONTRACT NUMBER HR0011-11-C-0041		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
				5d. PROJECT NUMBER		
6. AUTHOR(S) Jeffrey Draper Matthew French				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Southern California University Gardens, STE 203 Los Angeles, CA 90089-0001				8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA CMO 3701 N. Fairfax Drive Arlington, VA 22203-1714				10. SPONSOR/MONITOR'S ACRONYM(S)		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT Unlimited						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT <p>As commercial market forces are driving Integrated Circuit (IC) foundries offshore, the U.S. government is increasingly becoming concerned with the integrity of electronics procured from such offshore, uncontrolled facilities. Similarly, the government is intensely interested in the useful lifespan of these components. DARPA's Microelectronics Technology Office established the Integrity and Reliability in Integrated Circuits (IRIS) program to investigate methods of validating the functionality and reliability of ICs to address this issue. The Information Sciences Institute of the University of Southern California (USC/ISI) performed research in this area by supplying benchmark Test Articles (TAs) to better focus and drive the results of the IRIS program. USC/ISI has the unique blend of skills, IP, and resources, to not only develop and support each test article, but to do so in a cost-effective manner on State of the Art (SoA) process technologies</p>						
15. SUBJECT TERMS						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Jeffrey Draper	
U	U	U	UU	430	19b. TELEPHONE NUMBER (Include area code) 310-448-8750	

## Table of Contents

<b>1. Introduction .....</b>	<b>1</b>
<b>2. Phase 1 Test Article Description, Development, and History .....</b>	<b>2</b>
2.1 Technical Area 1 Test Articles .....	2
2.1.1 Thrust Area 1A: Non-destructive Analysis .....	2
2.1.2 Thrust Area 1B: Functional Derivation .....	3
2.2 Technical Area 3 Test Articles .....	4
2.2.1 Thrust Area 3A: ASIC IP.....	4
2.2.2 Thrust Area 3B: FPGA IP.....	5
2.3 Technical Area 4a Test Article .....	7
<b>3. Phase 2 Activities.....</b>	<b>9</b>
3.1 Reliability Characterization of Phase 1 Technical Area 4a Test Article .....	9
3.2 Advanced Techniques for Challenging ASIC Integrity/Reliability.....	21
3.3 Advanced Techniques for Challenging FPGA Integrity/Reliability.....	22
3.3.1 Stuck-at Fault Modeling and Testing for FPGAs .....	22
3.3.2 Discovery of Undocumented Functionality for FPGAs.....	24
<b>4. Appendix – Article Datasheets.....</b>	<b>25</b>
<b>5. Appendix – Phase 2 Digital ASIC Step-Stress and Lifetime Testing Results</b>	<b>26</b>

## 1. Introduction

As commercial market forces are driving Integrated Circuit (IC) foundries offshore, the U.S. government is increasingly becoming concerned with the integrity of electronics procured from such offshore, uncontrolled facilities. Similarly, the government is intensely interested in the useful lifespan of these components. DARPA's Microelectronics Technology Office established the Integrity and Reliability in Integrated Circuits (IRIS) program to investigate methods of validating the functionality and reliability of ICs to address this issue. The Information Sciences Institute of the University of Southern California (USC/ISI) proposed to aid the government in performing research in this area by supplying benchmark Test Articles (TAs) to better focus and drive the results of the IRIS program. USC/ISI has the unique blend of skills, IP, and resources, to not only develop and support each test article, but to do so in a cost-effective manner on State of the Art (SoA) process technologies.

Over the course of Phase 1 of the IRIS program, the ITAG (IRIS Test Article Generation) project delivered test articles for Technical Areas 1, 3, and 4a of the IRIS program. These test articles were comprised of ASIC hardware devices, ASIC design files, or FPGA design files, as mandated by the targeted Technical Area. At the government's direction, the test articles were delivered to the IRIS contractor community. From previous DARPA computer architecture projects, USC/ISI has a substantial base of open-source architecture designs which were leveraged to develop the test articles. USC/ISI had also developed FPGA CAD tools under DARPA and NASA efforts which can read, analyze, and modify FPGA design files at any point in the design process, which were extended to modify the circuits for testing detection capabilities. Both the architecture block IP and FPGA CAD tool IP represent significant previous investments for which the government has unlimited rights and saved the IRIS program substantial time and money. These technologies enabled the release of the Technical Area 4a article within eight months of the program start and continued to support later IRIS test articles for full use and redistribution within the IRIS program. The Technical Area 4a article was based on an existing RISC processor design. Subsequent test articles for other technical areas were scaled in size, complexity, and/or fabrication technology.

Due to sequestration and other budget cuts, the IRIS program redirected Phase 2 activities to explicitly focus on reliability issues and FPGA exploration activities. Much of the ASIC test article effort focused on detailed reliability characterization across a number of lots of the Phase 1 Technical Area 4a RISC processor chip.

USC/ISI also operates the MOSIS shared fabrication service, which was utilized under ITAG to aggregate designs on a dedicated IBM 9SF run through the TAPO program. By aggregating prototype and low-volume designs onto a single wafer, the substantial mask costs were shared over both Technical Area 4a and 4b test articles, leading to a significant cost savings for the U.S. government under this program.

Thus, the ITAG project played a vital strategic role in ensuring the success of the greater IRIS program and the awarded contractors and also contributed to the government's knowledge of the State of the Art (SoA) in assessing integrity and reliability vulnerabilities in ICs. This report serves as the final report for the IRIS (Integrity and Reliability in Integrated Circuits) Test Article Generation (ITAG) project. Thus, we focus this report on tasks performed and test articles developed by The University of Southern California's Information Sciences Institute (USC / ISI) in its role as the test article generation team for the program during both phases of the IRIS program.



## 2. Phase 1 Test Article Description, Development, and History

As noted above, it was vital to the IRIS program that a common set of benchmarks be developed to accurately evaluate each proposed approach, compare competing approaches, and select complementary approaches for end-to-end integration. The benchmarks proposed as a common platform for evaluating techniques which aim to assess integrity or reliability in custom chips consisted simply of various ASIC and FPGA Test Articles (TAs) developed under this ITAG effort. It is worth noting that although the performers in Thrust Areas 1 and 3 were not necessarily subscribed to find undocumented functionality in Phase I, these were included in the test articles for several reasons. First, this allowed the ITAG team to experiment with inserting undocumented features, thereby reducing Phase II yield risk and enabling a path for more sophisticated undocumented features in Phase II as well. This also gave the IRIS program a good indication of what kinds of undocumented features could be discovered with current techniques versus what would require DARPA level investigation. Finally, putting undocumented features into the Phase I articles allowed for them to be re-used as interim articles that performers could analyze during Phase II development before they took the final Phase II test.

More detail for articles for each Technical Area is given below.

### 2.1 Technical Area 1 Test Articles

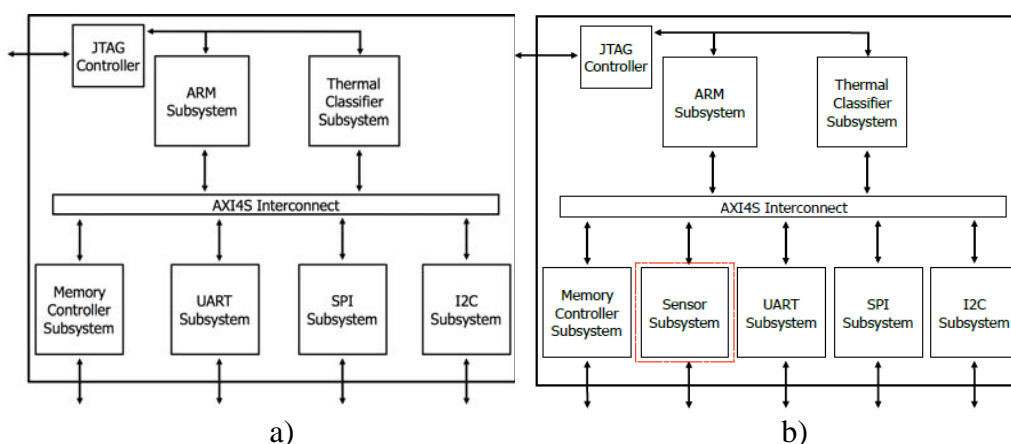
The goal of Technical Area 1 was to determine the functionality of an independently designed and fabricated IC in order to expose the presence of unwanted circuits. The test articles in this technical area served as benchmarks to measure the effectiveness of performer techniques in reverse engineering and processes to identify functionality of an IC. This technical area was subdivided into two classes Thrust 1A: Non-destructive Analysis and Thrust 1B: Functional Derivation, each of which required a unique test article. Thrust 1A focused on the non-destructive analysis of an IC in order to develop a flattened netlist design with sufficient detail to enable the derivation of an hierarchical netlist. Thrust 1B then focused on the next stage of deriving the hierarchical netlist and a detailed specification of the IC's functionality, given a netlist provided from area 1A. The test article for Thrust 1A was a fully packaged IC of approximately 1M transistors at 65nm, including a specification comparable to that provided in industry to end user's, and a representative test vector set in .vcd format. The test article for Thrust 1B was a flattened netlist of standard cells representing an approximately 1M transistor design at 65nm, specification comparable to that provided in industry to end user's, and a representative test vector set in .vcd format. It is important to note that the same design was not used in both articles in order to allow optimal testing of each thrust area's goals, and to ensure that performers that were awarded efforts in both 1A and 1B could not leak information across thrust boundaries. The following subsections provide a more detailed overview of the test articles that were developed for each thrust area.

#### 2.1.1 Thrust Area 1A: Non-destructive Analysis

The architecture selected for the test article of TA1A was a System on a Chip design representative of the image processing domain. The selection of this architecture allowed the testing of many different processing element types, interfaces, and programming models representative of DoD applications. As shown in Figure 1, this architecture consists of an ARM processor with custom circuitry to support hardware acceleration for hyperspectral imaging applications as well as several I/O and memory interfacing options, connected via a full-crossbar switch. From a design perspective, this diagram is really a collection of sub-systems, for which the performers are not given full description of and are expected to derive not just the top level diagrams but additional hierarchy as

well, especially within the ARM and AVR processor cores. Each subsystem can operate independently of the others and communicate over the AXI bus. Each subsystem operates at 100MHz. Full details of the test article are described in the data sheet which was provided to the performers, “ITAG Phase 1 Thrust 1A Test Article Datasheet,” and the answer key which was provided to the government team only, “ITAG Phase1 Thrust 1A Test Article Answer Key” These are included in the appendix for full reference.

In addition to the baseline design above, the ITAG team inserted several undocumented features to test the performer’s capabilities. The list of undocumented features can be found in the Errata List in the Answer Key document and includes: Unconnected Ring Oscillators, Health Monitoring Sensors, GSM Stream Cypher core, Performance Monitors, an extra I/O pin, extra ARM registers, writable UART Counters, and additional Memory Control Address pins. Most of these fall within the realm of items that an IC developer may not disclose to end users as they are either used for internal diagnostics, are features the IC developer chose not to support, or are errors the manufacturer did not wish to disclose. Further description of each of these undocumented features can be found in the Answer Key.



**Figure 1 TA1 Test Article Top Level SoC Diagram a) Performer b) Internal**

These test articles were designed and fabricated in the IBM 10LPE (65nm) process. The design was submitted for fabrication through TAPO on run 12A on March 1, 2012. Bare die were received on August 1, 2012, and parts were delivered to appropriate performers on schedule on August 30, 2012.

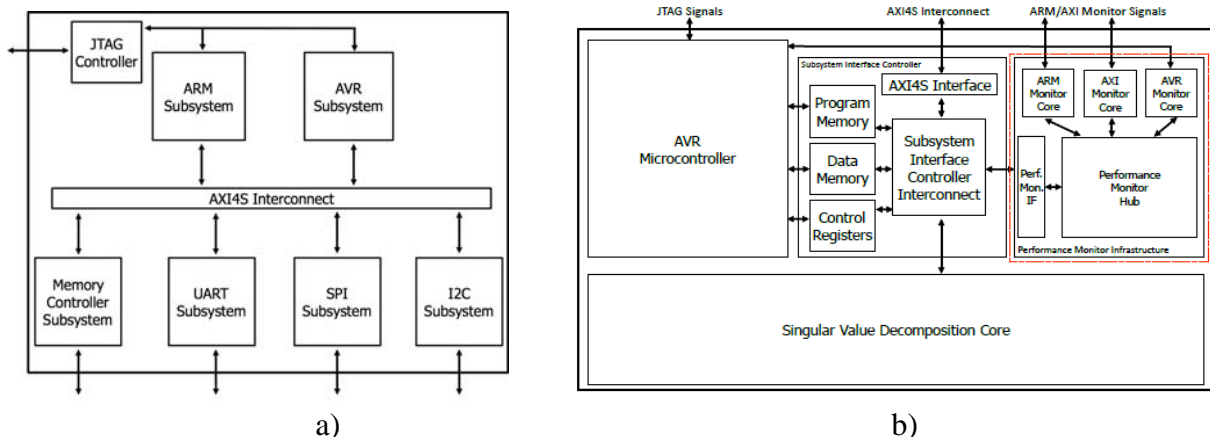
### 2.1.2 Thrust Area 1B: Functional Derivation

The architecture selected for the test article of TA1B was a System on a Chip design representative of the signal processing domain. The selection of this architecture allowed the testing of many different processing element types, interfaces, and programming models representative of DoD applications. As shown in X, this architecture consists of an ARM processor with custom circuitry to support hardware acceleration for Singular Value Decomposition (SVD) calculations as well as several I/O and memory interfacing options, connected via a full-crossbar switch. It is important to note that the implementation of several subsystems were different compared to TA1A, completely new subsystems were added, and some subsystems were removed. The ARM cache size was doubled from TA1A. The ABMBA AXI4 interconnect ports were reordered and the width was decreased to 16 bits. The SVD and VGA cores were added and the sensor core was removed. Several implementation steps such as artificially warping the system hierarchy, applying polymorphic func-

tional clusters, performing disjoint logic cell substitution, placing artificial cell restriction islands, and resynthesizing to different cell types were performed. As such, the resulting GDSII will look remarkably different than that of TA1A.

From a design perspective, this diagram is really a collection of sub-systems, for which the performers are not given full description of and are expected to derive. Each subsystem can operate independently of the others and communicate over the AXI bus. Each subsystem operates at 100MHz. Full detail of the test article is described in the data sheet which was provided to the performers, “ITAG Phase 1 Thrust 1B Test Article Datasheet,” and the answer key which was provided to the government team only, “ITAG Phase 1 Thrust 1B Test Article Answer Key.” These are included in the appendix for full reference.

In addition to the baseline design above, the ITAG team inserted several undocumented features to test the performer’s capabilities. The list of undocumented features can be found in the Errata List in the Answer Key document and includes: AXI interconnect port scheduling modification, enabling the ARM JTAG interface to read and write to the ARM program counter, inclusion of a GSM Stream Cypher core, inclusion of Performance Monitors, an extra I/O pin to support high resolution VGA, an extra I/O pin to change the ordering of the SVD results, and extra I/O pin to put the Memory Controller into pass-through mode, extra ARM registers, and additional Memory Control Address pins. Most of these fall within the realm of items that an IC developer may not disclose to end users as they are either used for internal diagnostics, are features the IC developer chose not to support, or are errors the manufacturer did not wish to disclose. Further description of each of these undocumented features can be found in the Answer Key.



**Figure 2 a) Disclosed Top-level Functionality of TA1B b) Detail of SVD subsystem including undocumented Performance Monitors (red)**

## 2.2 Technical Area 3 Test Articles

Technical Area 3 focused on determining the functionality of an independently designed functional block of digital IP integrated into the overall design of an ASIC or FPGA. This Technical Area was partitioned into Thrust 3A, which focused on ASIC soft IP delivered in human readable HDL, and Thrust 3B, FPGA IP delivered as a netlist. The following two subsections describe these two areas in more detail.

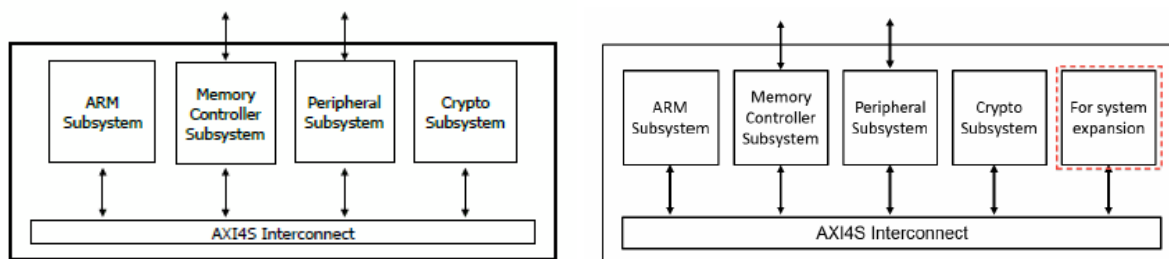
### 2.2.1 Thrust Area 3A: ASIC IP

The baseline design of the TA3A article is heterogenous multi-processor bus-based SoC targeting cryptographic hashing while supporting several hardware and software interfaces and programming models. The cryptographic features are provided through a coprocessor accelerator whose default algorithm can be switched at runtime. The user can operate these processors under stand-alone or parallel programming models. Additionally, TA3A provides multiple I/O and memory interfacing options, allowing a shared memory model, distributed memory model, or a hybrid shared-distributed memory model. These interfaces also enable the SoC to be used with other board- or system level devices. A special memory interface likewise allows external devices to push high-bandwidth data into the device, to provide an alternative mechanism to configure and control the device. The system's full-crossbar switch enables concurrent connectivity between subsystems to maximize on-chip communication bandwidth. These features make TA3A a specialized and flexible processor for cryptographic applications.

TA3A is internally composed of multiple subsystems connected through an AXI4S bus. The subsystems include an ARM processor, a ZPU processor with a cryptographic accelerator, a memory controller, and a UART interface. The architecture of this SoC allows each subsystem to function independently, with its own dedicated AXI4S port and reset signal. The ARM and Crypto subsystems are masters on the AXI bus. The UART is a slave on the AXI bus and must be polled for incoming data. The Memory subsystem allows the system to access off-chip memory. Each of these subsystems operates at the system clock speed. The high level block diagram is shown in Figure 3.

In addition to the baseline design above, the ITAG team inserted several undocumented features to test the performer's capabilities. The list of undocumented features can be found in the Errata List in the Answer Key document and includes: An extra AXI4S interconnect port, a bypass mode to the cryptographic subsystem which allows the entire encryption module to be disabled, modification of the routing arbitration algorithm to be biased to higher port numbers, insertion of system performance monitors, making the originally read only UART counters to be writeable, and a mismatch in the address pins between the system and the memory controllers.

Full detail of the test article is described in the data sheet which was provided to the performers, "ITAG Phase 1 Thrust 3A Test Article Datasheet," and the answer key which was provided to the government team only, "ITAG Phase 1 Thrust 3A Test Article Answer Key." These are included in the appendix for full reference.



**Figure 3 TA3A a) baseline block diagram b) block diagram of modified system.**

## 2.2.2 Thrust Area 3B: FPGA IP

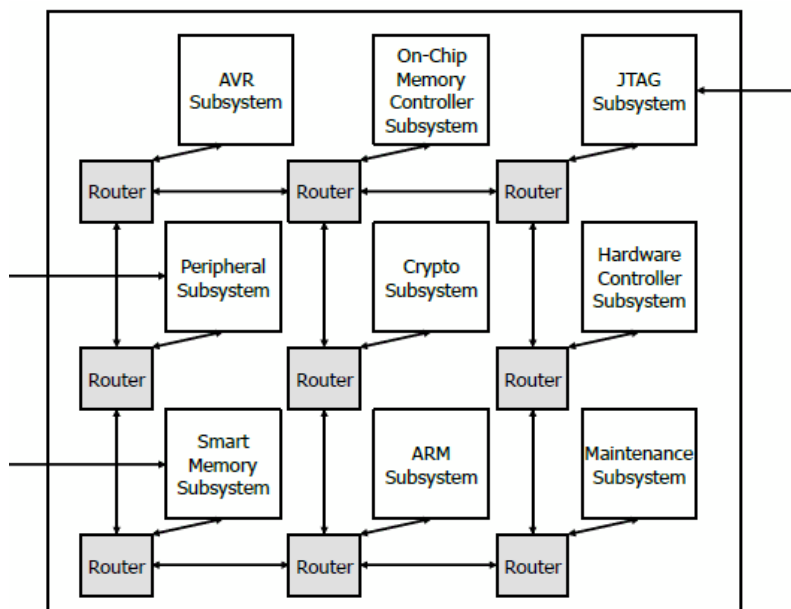
The baseline design of the Thrust 3B Test Article (TA3B) is a soft IP System-on-Chip (SoC) intended for implementation in Xilinx Virtex6 and Virtex7 FPGAs. The TA3B architecture includes a heterogeneous multi-processor on-chip mesh network-based SoC targeting cryptographic hashing while supporting several hardware and software interfaces and programming models. The cryptographic features are provided through a coprocessor accelerator whose default algorithm can be switched at runtime. The user can operate these processors under stand-alone or parallel programming models. Additionally, TA3B provides multiple I/O and memory interfacing options, allowing a shared memory model, distributed memory model, or a hybrid shared-distributed memory model. These interfaces also enable the SoC to be used with other board- or system-level devices. The system's on-chip mesh network enables concurrent connectivity between subsystems to maximize on-chip communication bandwidth. These features make TA3B a specialized and flexible processor for cryptographic applications.

TA3B is internally composed of multiple subsystems connected through an AXI4S mesh on-chip network. The subsystems include an ARM processor, two AVR processors, and a ZPU processor with a cryptographic accelerator that provides two SHA-3 candidates. One AVR processor is used for system maintenance while the second is available for power-efficient processing. The system also includes a memory controller, a hardware control core, a JTAG interface, and peripheral interfaces with UART, timers, and interrupt controller. The ARM, AVR, and ZPU processor subsystems are masters on the AXI4S on-chip network. The UART, timer and interrupt controller are AXI4S slaves and must be polled for incoming data. The Memory subsystem gives the system access to off-chip memory. Each of these subsystems operates at the system clock speed.

In addition to the baseline design above, the ITAG team inserted several undocumented features to test the performer's capabilities. The list of undocumented features can be found in the Errata List in the Answer Key document and includes: an undocumented GSM A5/1 stream cypher core attached to the ARM coprocessor, support for runtime reconfiguration of the AXI4 mesh interconnect, reduction of the data width of the mesh network from 32 to 16 bits wide, connection of the ZPU processor's data memory to the JTAG chain, insertion of system performance monitors, UART registers modified from read only to write, insertion of a bypass mode into the cryptographic subsystem to circumvent the SHA-3 hash function, an undocumented mode of the cryptographic subsystem which implements the Skein hash function, and a mismatch in the number of pins between the memory controller and the system.

This article was delivered as synthesizable, human readable HDL (both Verilog and VHDL) with datasheet and test vector set in .vcd format. Figure 4 depicts the high level block diagram of the TA3B system. Full detail of the test article is described in the data sheet which was provided to the performers, "ITAG Phase 1 Thrust 3A Test Article Datasheet," and the answer key which was provided to the government team only, "ITAG Phase 1 Thrust 3A Test Article Answer Key." These are included in the appendix for full reference.



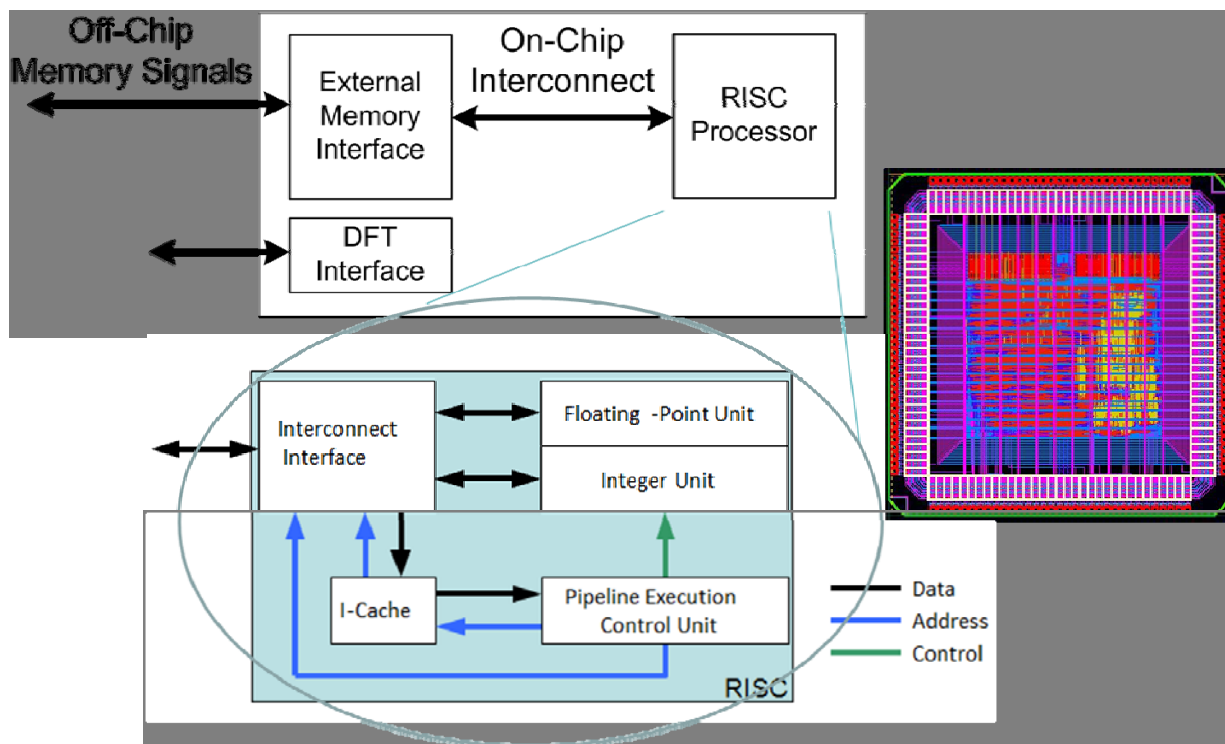


**Figure 4 TA3B Base Design**

### 2.3 Technical Area 4a Test Article

As described in DARPA BAA DARPA-BAA-10-33 for the IRIS program, the goal of Technical Area 4 was the development of innovative concepts for assessing the reliability of a batch of ICs based on testing of very small numbers (~10) of ICs and, ideally, the ability to assess nondestructively the expected reliability of a single IC. The focus of Technical Area 4a was Digital Reliability. Reliability screening techniques were expected to ideally address a full range of physics of failure expected for current and advanced CMOS process nodes (e.g. 45 nm or below) and be able to identify ICs with potential reliability problems, whether caused by normal statistical variations, manufacturing quality issues, or even intentional tampering.

The TA4AP1 (internal code name of ITAGR1) test article developed for this technical area contains a RISC processor connected to an external memory interface through a point-to-point interconnect. The block diagram of the RISC processor with respect to the interconnect and the external memory interface along with an image of the layout is shown in Figure 1. Since the IRIS program schedule called for the first article delivery for this technical article very early in the program, the ITAG project leveraged a design from the DARPA Trust in IC program that was called TA2 Software Article, with one notable exception. The memory interface of ITAGR1 has been redesigned to transform memory accesses into a burst of 32-bit transfers to reduce the pad/pin count of the resulting design. The point-to-point interconnect is implemented by the node bus interface (or memory interface) of each RISC processor. Besides serving as a controller for an external memory system, the external memory interface contains a node bus interface for interaction with the RISC processor. More detailed information about this test article can be found in the IRIS Test Article 4A Phase 1 (TA4AP1) Datasheet in the appendix along with accompanying documents Test Article 2 Software Article RISC Processor Architecture Overview, Test Article 2 Software Article RISC Processor Instruction Set Manual, and Test Article 2 Software Article Memory Interface Description.



**Figure 2.3-1: TA4AP1 Block Diagram and Implementation Layout**

As noted in the datasheet, the design was implemented in IBM 9SF technology and contained around 1.4 million transistors. A brief history of the test article development is given below:

- Taped out late September 2011
- Released to Manufacturing early November 2011
- Parts back from fab mid January 2012
- Packaging February 2012
- Pass-fail lot sorting based on worst-case pre-fabrication simulation speed conducted March 2012
- POR parts distributed to performers August through October 2012 upon request

With process and design variations (detailed in separate sensitive documentation), there were a total of 12 different lots of chip types. The delivery log for the TA4AP1 test article is shown below.

Quantity	Form	Lot Type	Date Delivered	Recipient
All	PGA132	All	1-Jul-12	TestEdge (all parts returned upon sorting)
10	PGA132	POR	30-Aug-12	Boeing - Ethan Cannon (parts returned and reissued to IBM/Peilin Song 10/18/2)
10	PGA132	POR	19-Sep-12	IBM - Peilin Song
10	bare die	POR	19-Sep-12	IBM - Peilin Song
10	PGA132	POR	19-Sep-12	Georgia Tech - Linda Milor
10	PGA132	POR	29-Oct-12	ISI - Mike Bajura
2	PGA132	POR	20-Feb-13	DMEA - Daniel Marrujo
4	PGA132	POR	21-Feb-13	Aerospace - Jon Osborn
2	PGA132	POR	15-May-13	Crane - Brett Hamilton
6	bare die	POR	15-May-13	Crane - Brett Hamilton
10	bare die	POR	29-May-13	SRI - David Stoker
8	PGA132	POR	3-Jul-13	Aerospace - Jon Osborn
5	bare die	POR	10-Sep-13	SRI - David Stoker
6	PGA132	POR	17-Sep-13	DMEA - Daniel Marrujo
21	PGA132	POR	27-Sep-13	TestEdge (parts for step-stress testing)
20	bare die	POR	9-Oct-13	Raytheon/ASI - Erika Clausen
10	bare die	POR	23-Oct-13	SRI - David Stoker
16	PGA132	S9	13-Nov-13	TestEdge for step-stress test
210	PGA132	POR	27-Jan-14	DMEA - Daniel Marrujo for life test
189	PGA132	S9	28-Jan-14	DMEA - Daniel Marrujo for life test
189	PGA132	S3	31-Jan-14	DMEA - Daniel Marrujo for life test
2,2	PGA132	S3, S9	31-Jan-14	IBM - Peilin Song
1,1,1,1,1	PGA132	S0,S1,S3,S4,S9	7-Feb-14	IBM - Peilin Song (stressed parts)
1,1	PGA132	S1, S4	7-Feb-14	IBM - Peilin Song
2	PGA132	S6	28-Feb-14	IBM - Peilin Song
10	bare die	S9	28-Feb-14	BAE Systems - Daniel S. Pineo
8	PGA132	S9	28-Feb-14	ISI - Mike Bajura
2	bare die	S9	28-Feb-14	ISI - Mike Bajura
10	bare die	S9	28-Feb-14	Raytheon/Micronet - Erika Clausen
10	bare die	S9	28-Feb-14	SRI - David Stoker
2,2,2	PGA132	S8,S10,S11	8-Apr-14	IBM - Peilin Song
10	bare die	S9	18-Jun-14	Raytheon/Micronet - Erika Clausen
15	bare die	S9	25-Jun-14	Raytheon/Micronet - Erika Clausen

### 3. Phase 2 Activities

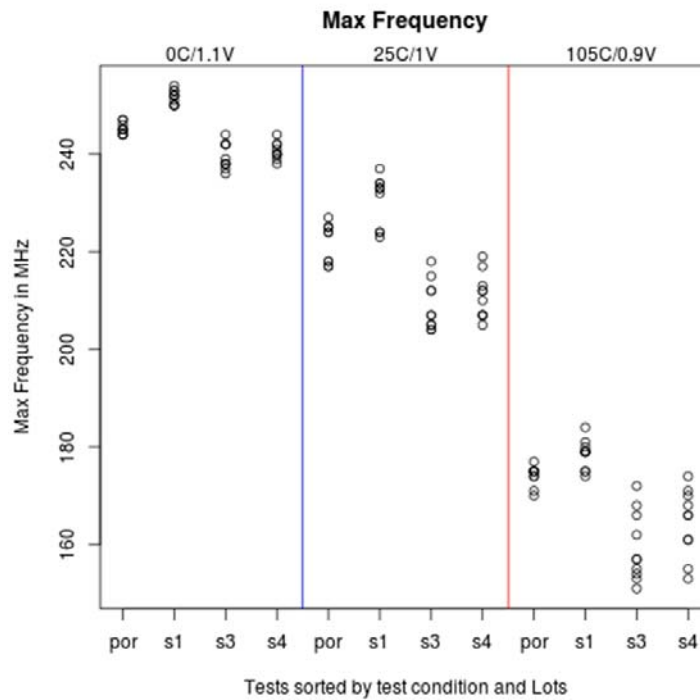
As noted in the introduction, the IRIS program redirected Phase 2 activities to explicitly focus on reliability issues and exploration activities. Much of the ASIC test article effort focused on detailed reliability characterization across a number of lots of the Phase 1 Technical Area 4a RISC processor chip and developing an exploration test article largely derived from the Phase 1 Technical Area 1a test article. Similarly, activities to explore the use of advanced techniques for challenging integrity and/or reliability issues in FPGA designs were conducted.

#### 3.1 Reliability Characterization of Phase 1 Technical Area 4a Test Article

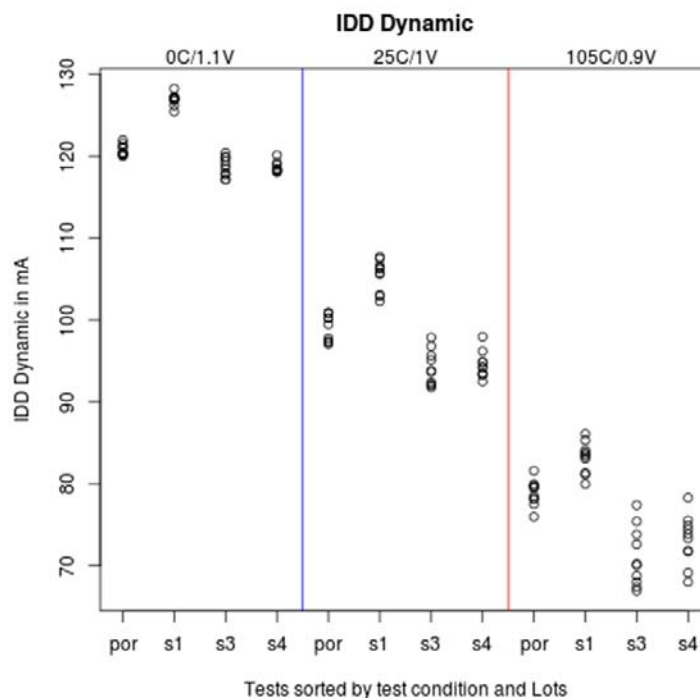
Given the DARPA IRIS program re-direct of Phase 2 to focus more on the reliability assessment capabilities for limited lot sizes of ICs, a decision was to use variants of the TA4AP1 test article for further exploration. The ITAG team in conjunction with colleagues at DMEA and Aerospace formulated a plan for conducting thorough electrical characterization tests at voltage and temperature corners, step-stress tests for lots of interest, and lifetime testing for lots of interest to develop expectations for what performers would report on the program as part of their findings when assessing their proposed reliability prediction techniques. Recall that there were a total of 12 lot types of TA4AP1 arising from a combination of design and fabrication alterations. Since Phase 1 activities focused on simple pass-fail sorting of devices, the first major effort in Phase 2 focused on gathering more electrical characterization data for the 6 lot types that contained the baseline design. The performance metrics measured across voltage and temperature corners included fmax (maximum attainable operating frequency), static and dynamic current (IDD) for both the core and I/O, tpd (out-

“USE OR DISCLOSURE OF DATA CONTAINED ON THIS SHEET IS SUBJECT TO THE RESTRICTION ON THE TITLE PAGE OF THIS DOCUMENT”

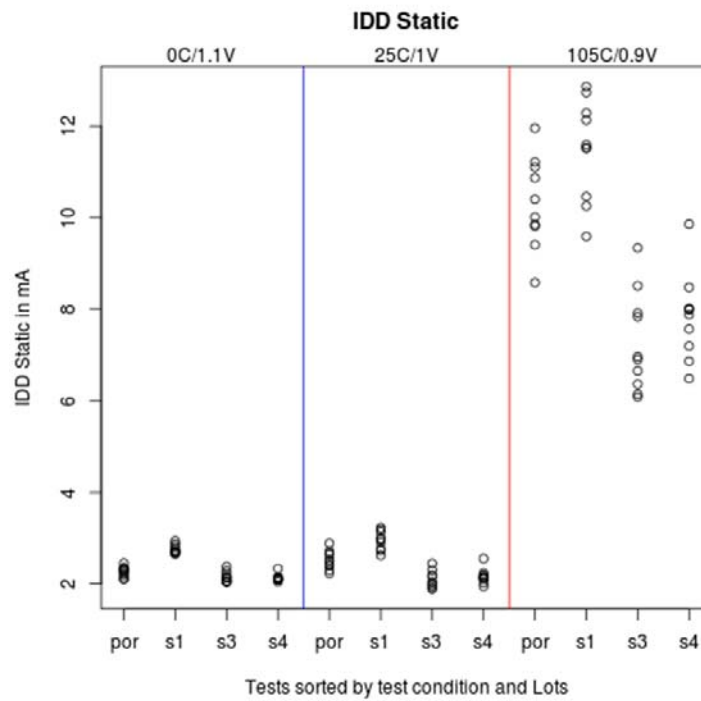
put propagation delay), ts (input setup time), th (input hold time). The next 8 figures present the data measure for 10-chip lots for each of the lot types POR (process of record), S1, S3, and S4.



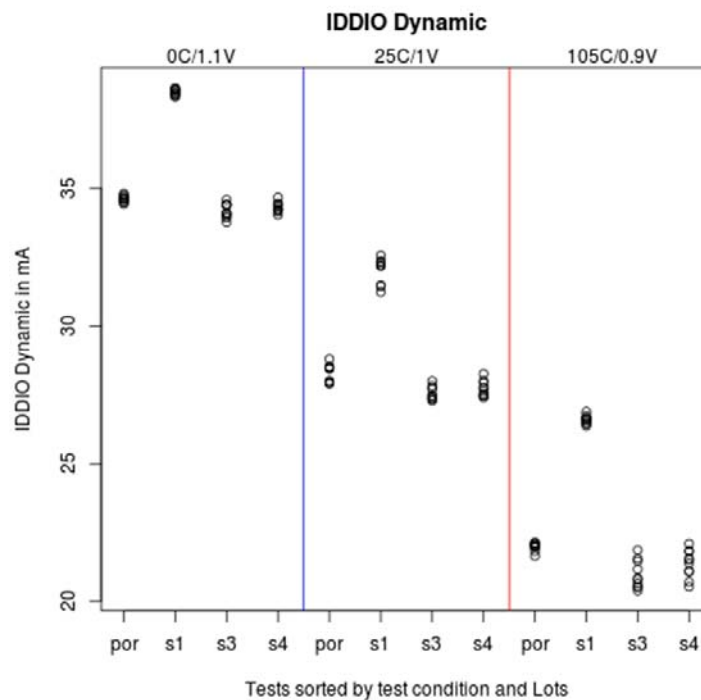
**Figure 3.1-1: Maximum Operating Frequency**



**Figure 3.1-2: Dynamic Core IDD**

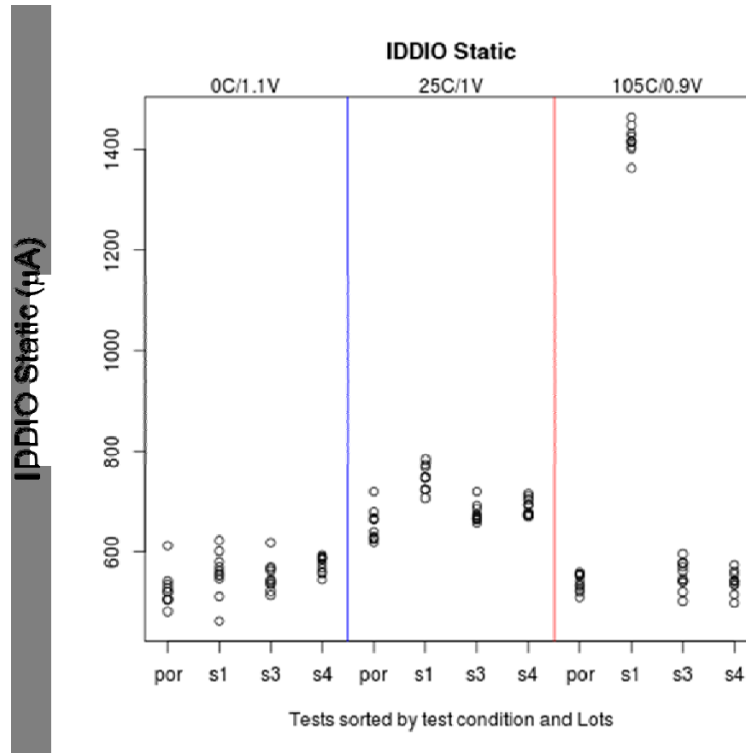


**Figure 3.1-3: Static Core IDD**

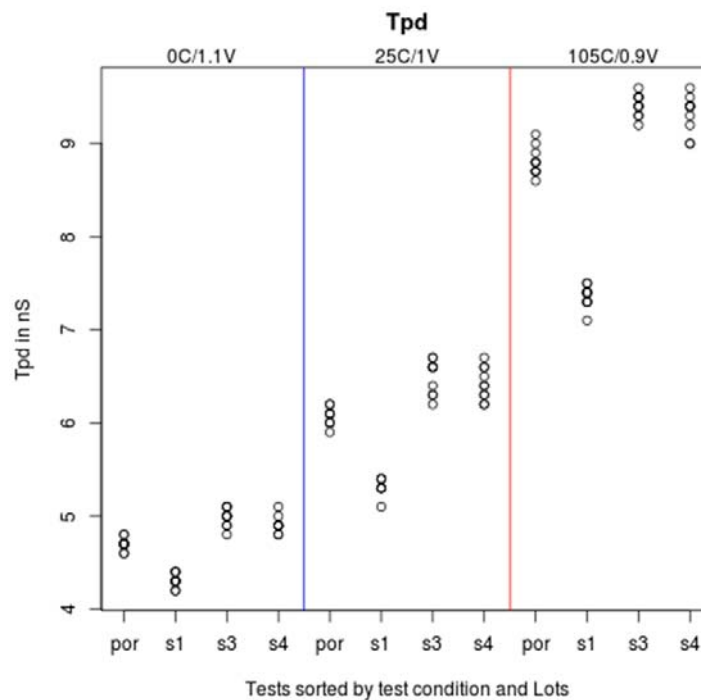


**Figure 3.1-4: Dynamic I/O IDD**

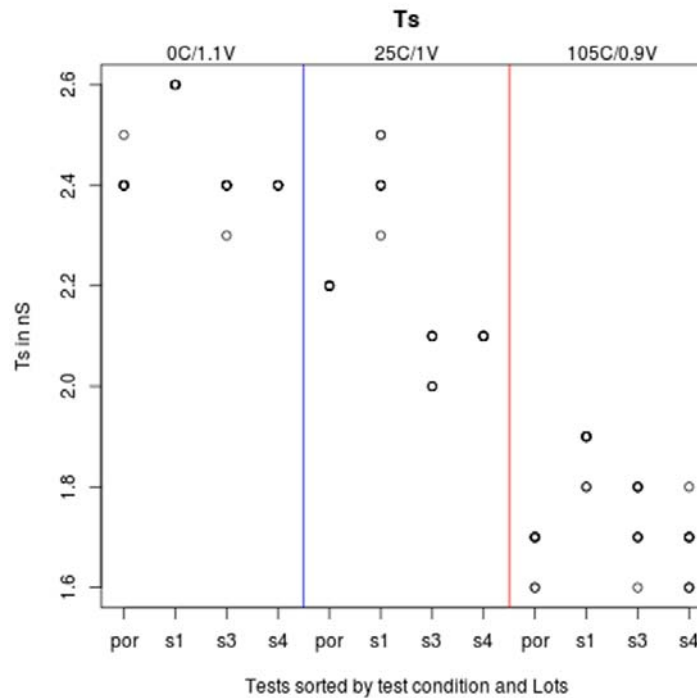




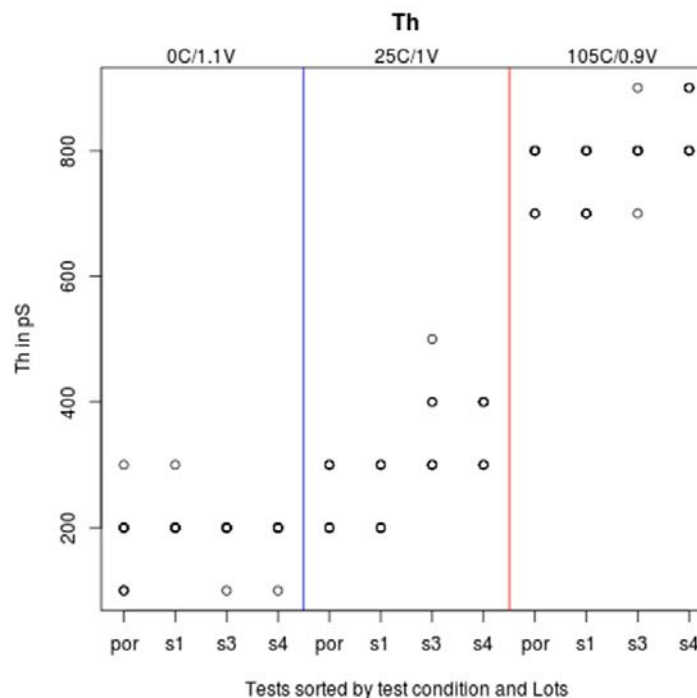
**Figure 3.1-5: Static I/O IDD**



**Figure 3.1-6: Output Propagation Delay (Tpd)**

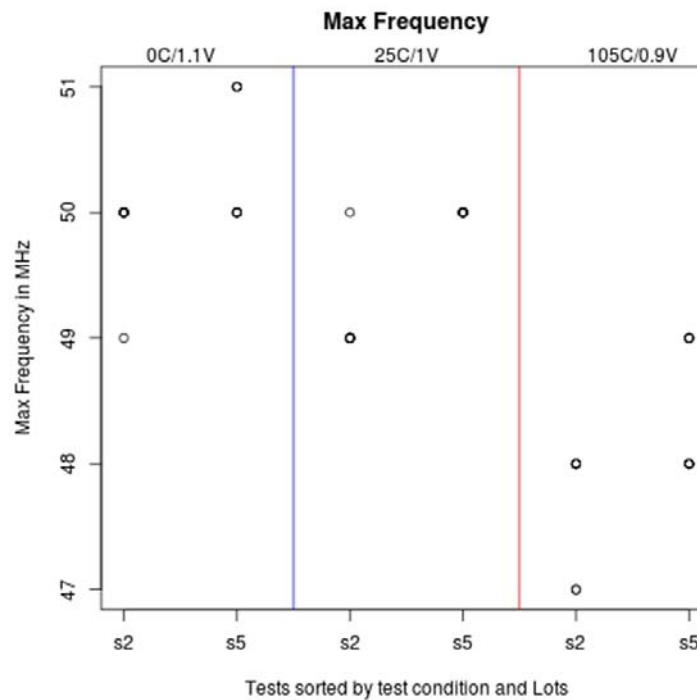


**Figure 3.1-7: Input Setup Time (Ts)**

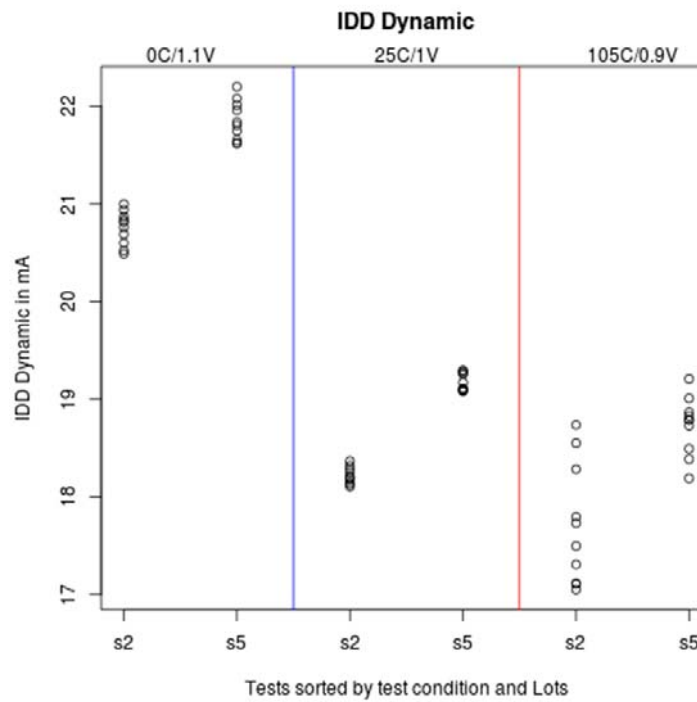


**Figure 3.1-8: Input Hold Time (Th)**

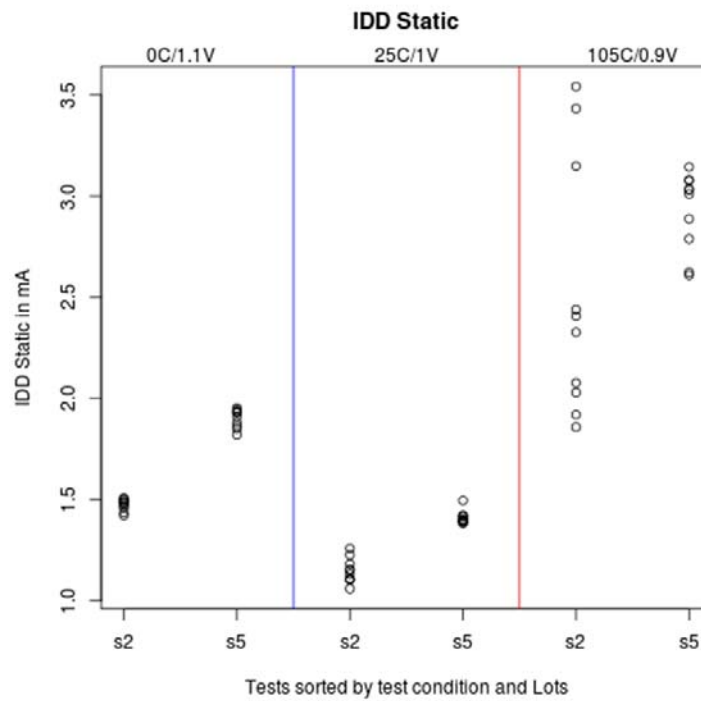
Another set of eight figures with the same parameters for two other lots is shown below. These lots are shown separately because of their severely limited operating conditions of running with the instruction cache turned off and much lower achievable maximum operating frequency.



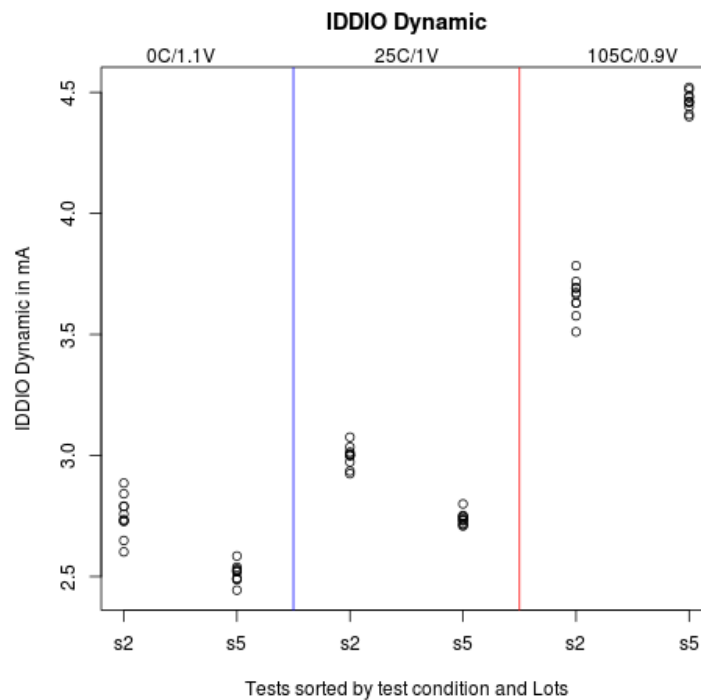
**Figure 3.1-9: Maximum Operating Frequency**



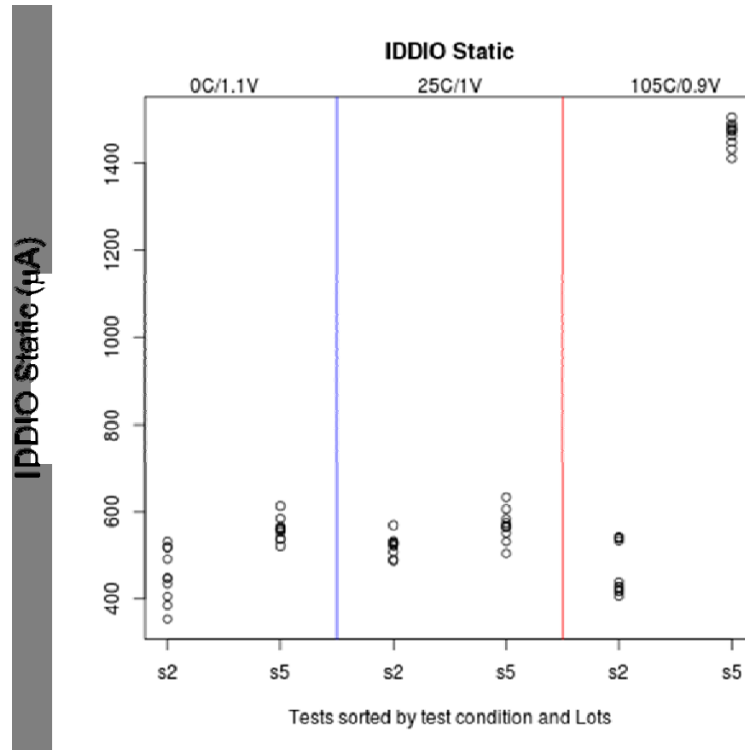
**Figure 3.1-10: Dynamic Core IDD**



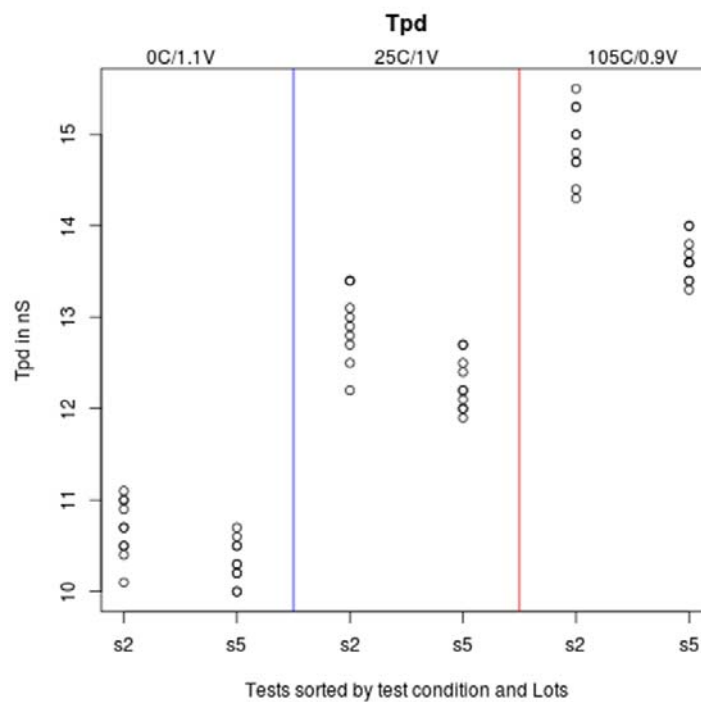
**Figure 3.1-11: Static Core IDD**



**Figure 3.1-12: Dynamic I/O IDD**

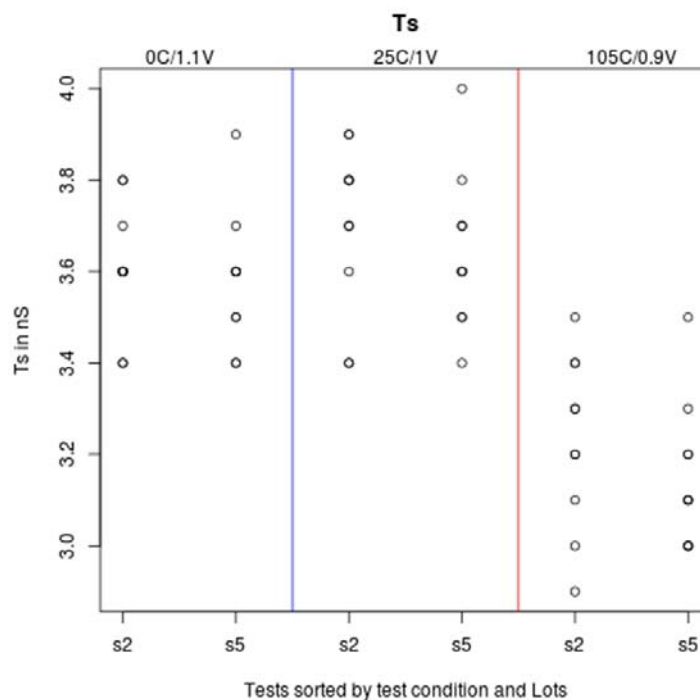


**Figure 3.1-13: Static I/O IDD**

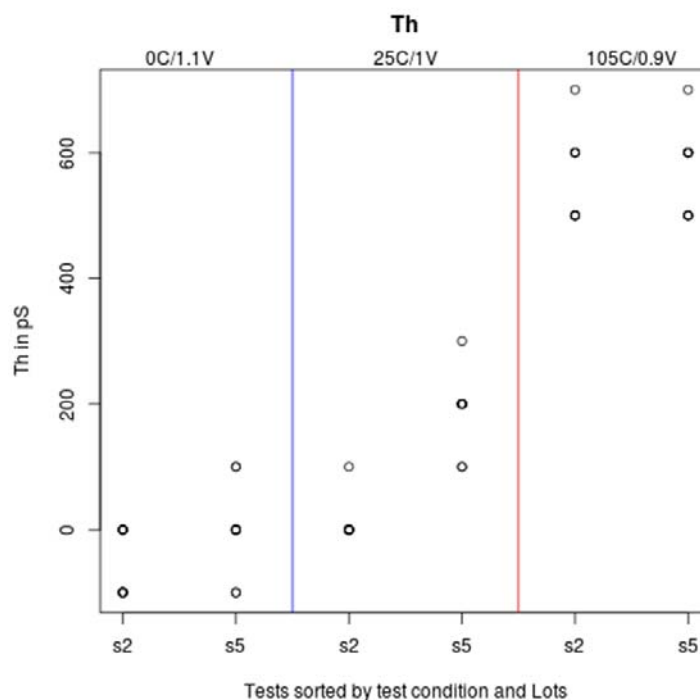


**Figure 3.1-14: Output Propagation Delay (Tpd)**



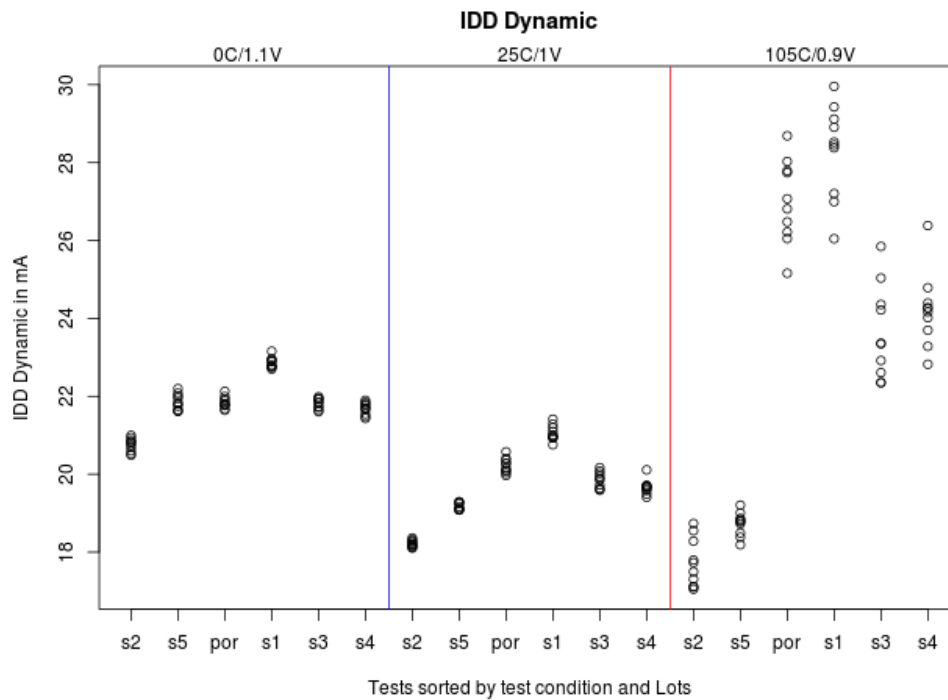


**Figure 3.1-15: Input Setup Time (Ts)**

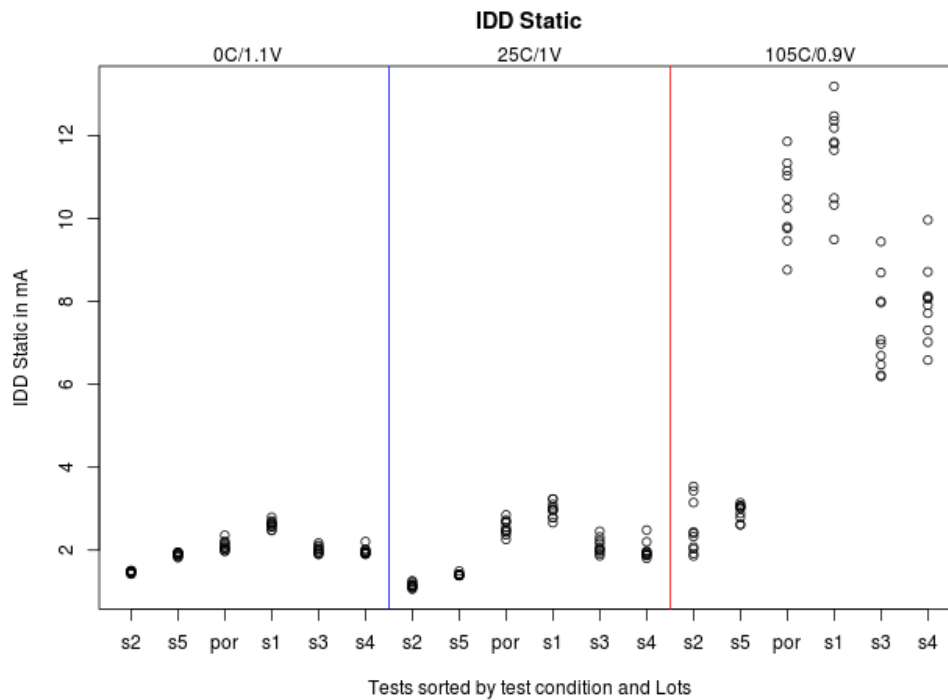


**Figure 3.1-16: Input Hold Time (Th)**

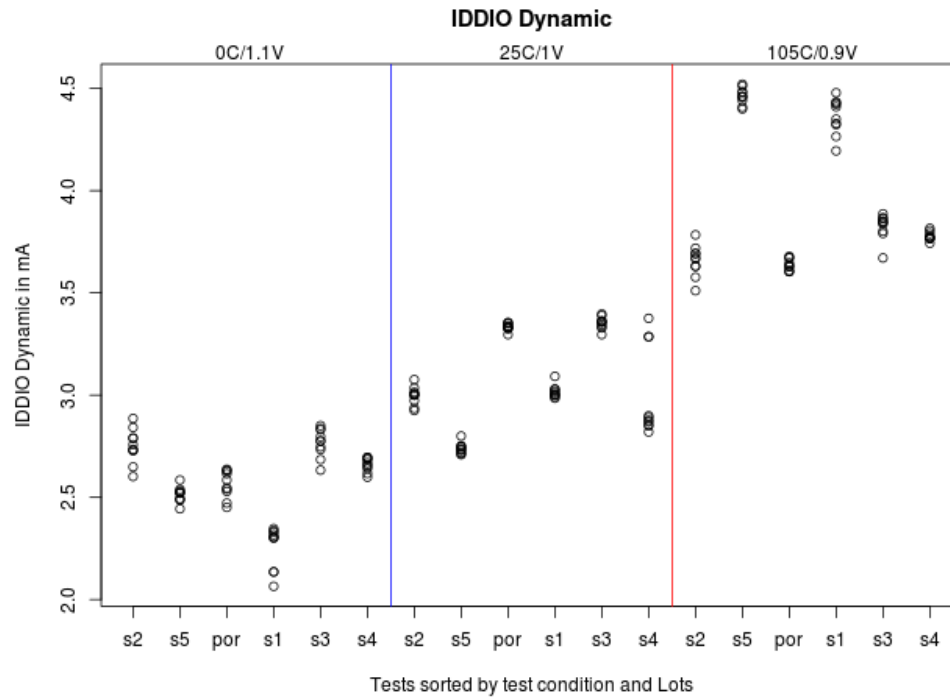
Since all prior data was taken at the maximum operating frequency of each chip, it was difficult to do some comparison across all lots, so one final set of data was taken for all lots at 50 MHz (the max operating frequency of lots s2 and s5) with the instruction cache off to better facilitate lot-to-lot comparisons. The resulting figures are shown below.



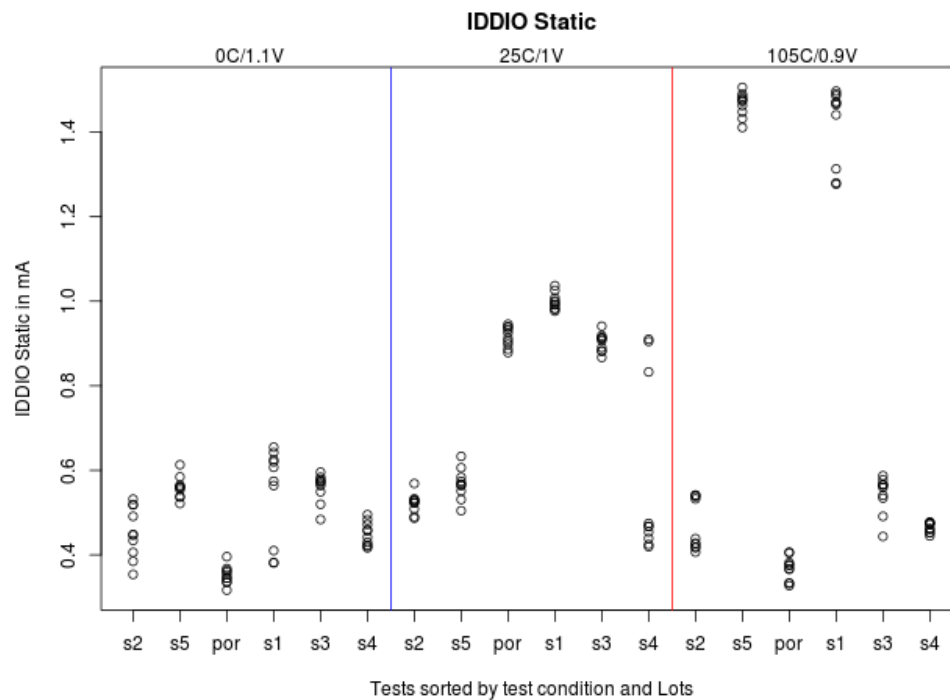
**Figure 3.1-17: Dynamic Core IDD at 50MHz with I-Cache Off**



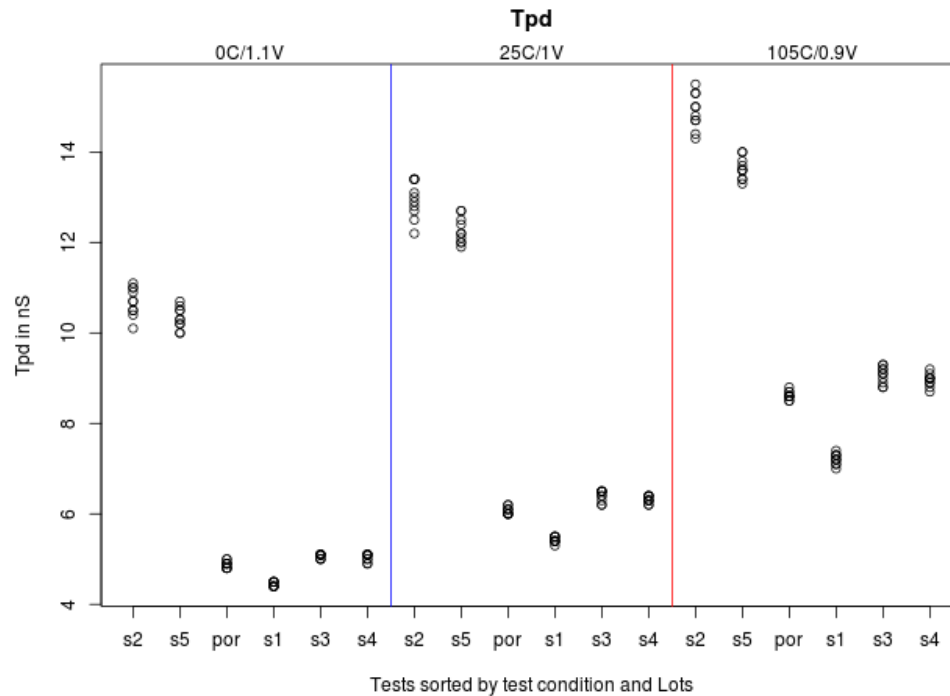
**Figure 3.1-18: Static Core IDD at 50MHz with I-Cache Off**



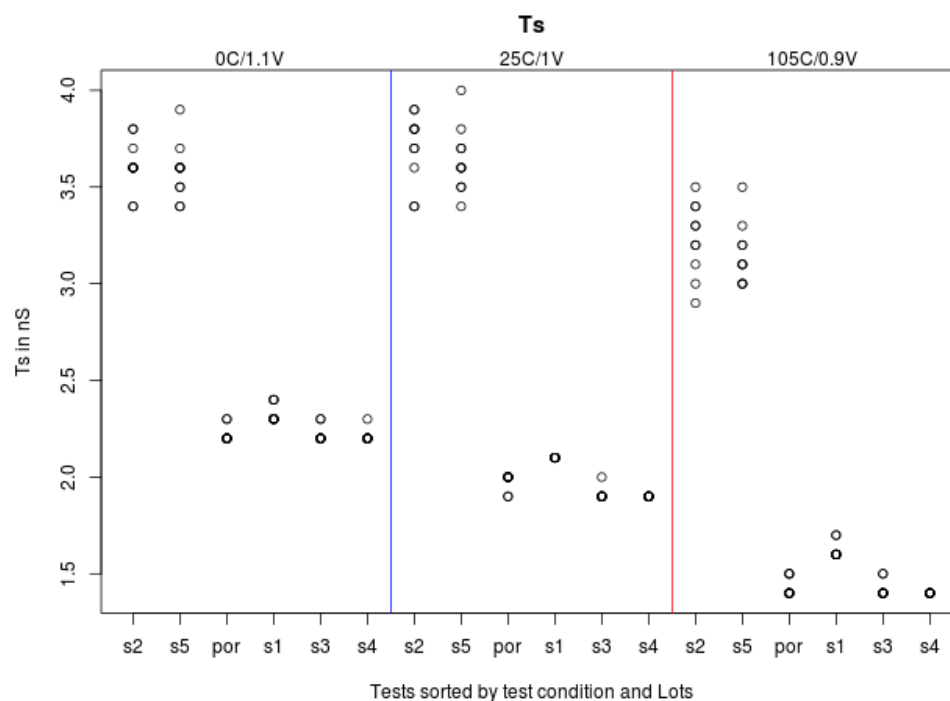
**Figure 3.1-19: Dynamic I/O IDD at 50MHz with I-Cache Off**



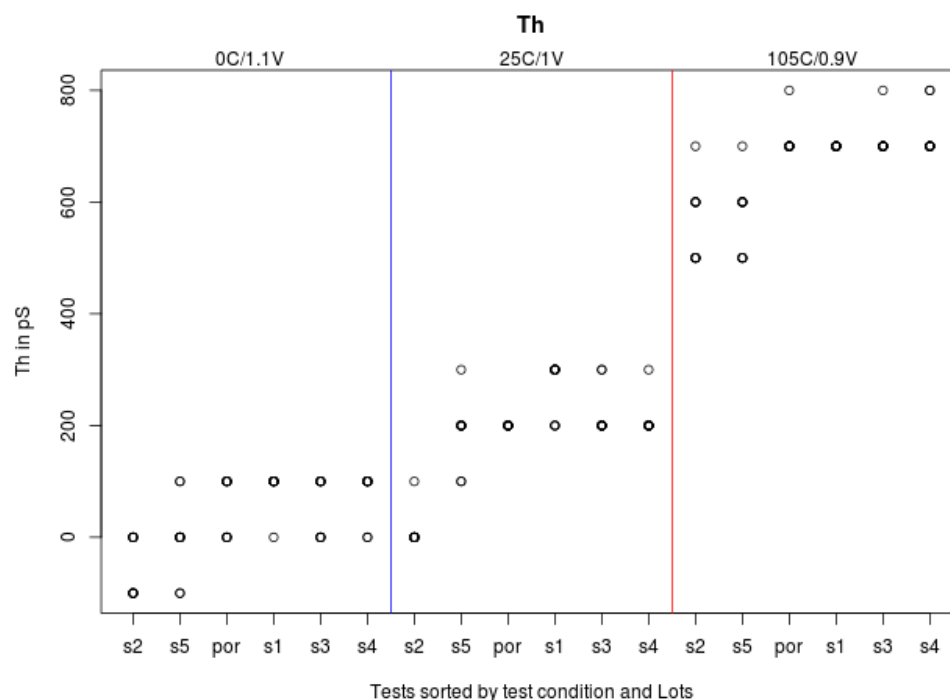
**Figure 3.1-20: Static I/O IDD at 50MHz with I-Cache Off**



**Figure 3.1-21: Output Propagation Delay (Tpd) at 50MHz with I-Cache Off**



**Figure 3.1-22: Input Setup Time (Ts) at 50MHz with I-Cache Off**



**Figure 3.1-23: Input Hold Time (Th) at 50MHz with I-Cache Off**

The original focus was on the baseline design, which applied to lots S0 (also known as POR) through S5. After initial step-stress testing, the emphasis was shifted to lot S9, an altered version of the design implemented on fabrication on lot S3. Some general conclusions from all the voltage-temperature testing that was performed include:

- Lots S2 and S5 operate only with cache off
  - Also lower max operating frequency (~1/4 that of other lots)
- Lots S3 and S4 have slightly lower max operating frequencies than POR
- Main distinction of S1 is I/O speed and current
- Main distinction between S3 and S9 is I/O static current

Once thorough electrical testing had been performed for most of the lots of interest, the focus shifted to step-stress testing and lifetime testing. This work was primarily performed by Aerospace under a subcontract to USC and DMEA. The Aerospace final report detailing this work is included in Appendix 5.

Parts from the S9 lot were delivered to performers for their final Phase 2 activities March 1, 2014.

### 3.2 Advanced Techniques for Challenging ASIC Integrity/Reliability

Additional Phase 2 activities involved the development of an ASIC test article to explore the use of advanced techniques for challenging integrity and/or reliability detection. The task used the Phase 1 Technical Area 1 test article fabricated in IBM 10LPe technology as a baseline design for developing the techniques and fabricated a chip in IBM 10RFe to support demonstration of the techniques. The IRIS government team provided input for the types of challenge circuits to be inserted into the baseline circuitry, and descriptions of the challenge circuitry were provided earlier through separate



sensitive documentation. The test article design taped out September 23, 2013, and the resulting fabricated chip was delivered to select government partners March 15, 2014.

### **3.3 Advanced Techniques for Challenging FPGA Integrity/Reliability**

Under this task, USC/ISI explored the use of advanced techniques for challenging integrity and/or reliability issues in FPGA design in the areas of stuck at fault modeling of the Xilinx Zynq slices and exploring undocumented functionality within the Xilinx Virtex 5 DSP48 hard IP module.

#### **3.3.1 Stuck-at Fault Modeling and Testing for FPGAs**

Functional testing of commercial FPGAs, independent of in-house FPGA vendor production testing, is an important first step in establishing a trusted supply-chain, determining the usability of devices stored in inventory for long periods of time, and for determining the health status of fielded systems. While current and next-generation FPGAs are increasingly using emerging technology to thwart counterfeiting attempts, older FPGA generations are easily recycled and sold as new. Devices in deep storage may not have been stored properly, and devices under heavy use or in strenuous operating environments may experience wear out effects. Independent functional testing of the FPGA VLSI provides a sanity check that the device is in fact the device it claims to be and is in good working order. This is no trivial feat as modern FPGA devices now contain over 1B transistors, over a dozen types of Hard IP, 35M user wires, and 380M user routing switches.

To address this, USC/ISI developed Independent FPGA Functional Testing (IFT) Tools which generate independent tests that can be used to cross-check the FPGA manufacturer's testing and can also be used for field testing of counterfeit, damaged, or aging parts. The ability to develop such tests relies upon exhaustive knowledge of the internal FPGA architecture. IFT provides such knowledge for all Xilinx FPGAs dating back to the original Virtex series, and allows automation of the test generation process. IFT currently supports the Xilinx 7-Series architectures (Virtex7, Kintex7, Artix7, Zynq700). Additional architectures can be added with a simple one-time porting effort. Support for any given architecture includes all devices within that architecture.

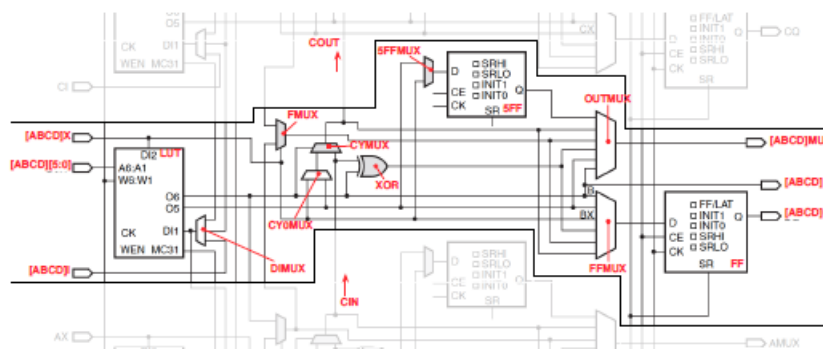
The IFT technical approach is to utilize these databases to generate test bitstreams which are loaded onto the FPGA under test. An on chip controller then exercises the test bitstreams to validate that the underlying VLSI of the device is working as expected. Our in-circuit testing approach assumes that the FPGA Device Under Test (DUT) is mounted on a PCB, and that special test access to external FPGA I/O pins is not available. This precludes the use of clock, reset, control, and monitoring signals. Other testing efforts in published literature do not accommodate these same restrictions. Required testing connectivity for IFT consists solely of power and an interface to the device Configuration Controller—either JTAG or SelectMAP.

The test bitstreams are carefully constructed to yield exhaustive coverage of the device, while also testing as many features in parallel in order to minimize testing time. For this effort, IFT provides testing coverage for SLICELs and SLICEMs as well as the routing of the devices. For the Zynq XC7Z020, there are a total of 24,240 logic sites of 88 different types. 13,300 of those are slices, 8,810 are power sources, and the remaining 2,130 are an assortment of DSPs, BRAMs, clock logic, high-speed transceivers, and other logic. By covering the slices and power sources, we achieve 91%

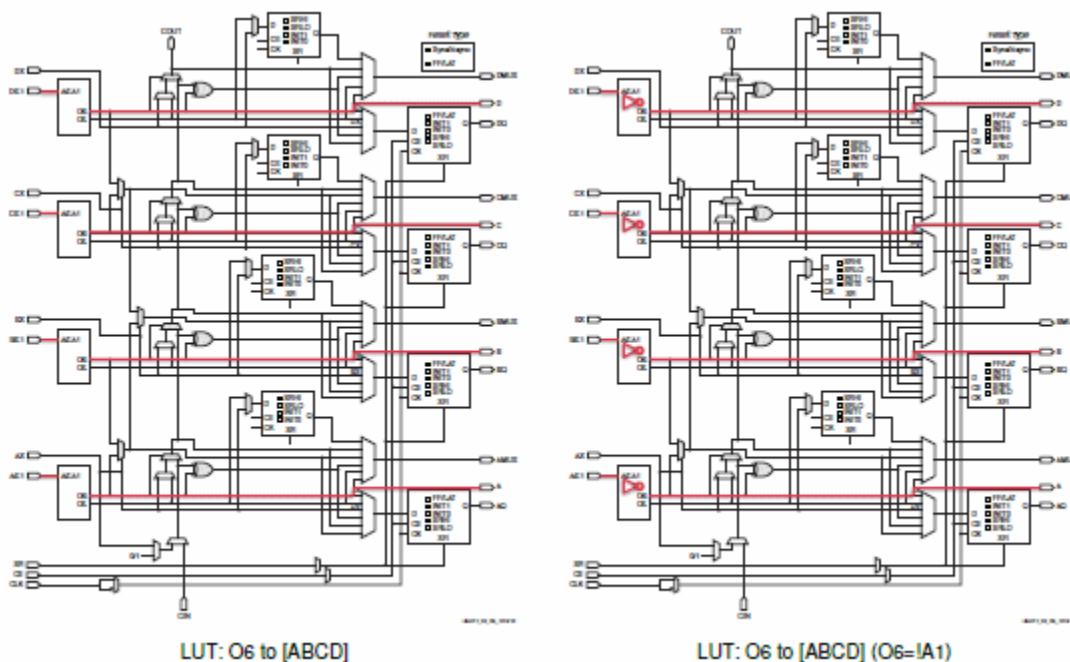
“USE OR DISCLOSURE OF DATA CONTAINED ON THIS SHEET IS SUBJECT TO THE RESTRICTION ON THE TITLE PAGE OF THIS DOCUMENT”

coverage of logic sites in this device. Routing tests leverage the interior tiles of the FPGA design, and currently provide 95% routing coverage. The percent coverage for both logic and routing increases for larger devices, because they contain a larger percentage of slices and interior routing tiles.

Creation of the proper bitstreams is an exacting task as not only does each logic site need to be tested, but also each path through a Slice must be validated and each path through the global routing Programmable Interconnect Points, must also be tested. Figure 5 shows all of the logic sites that are tested in our approach. Figure 6 shows two different test configurations that exercise two slightly different paths within a Slice.



### Figure 5 Slice Logic Sites



### Figure 6 Examples of Slice Path Test Bitstreams

Overall, the IFT tool is the first known comprehensive stuck at fault testing tool for FPGA devices. For further detail, please reference the ITAG Independent Functional Testing Tool Manual provided in the appendix.

### **3.3.2 Discovery of Undocumented Functionality for FPGAs**

The exploration of undocumented functionality was conducted for the DSP48E hard IP in the Xilinx Virtex 5 series FPGA. The DSP48E is one of the most commonly used hard IP blocks, represents 62% of the hard IP blocks in the V5 LX-110T device, and has a non-trivial number of control inputs and configuration settings to investigate. A manual inspection of the user guide documentation has discovered over 750 undocumented modes involving control inputs and configuration settings for the DSP48E. The DSP48E has been present on FPGAs since the Virtex 2 series and has undergone minor incremental changes in each new generation. There are 64 DSP hard IP blocks on the aforementioned Virtex5 FPGA, which enables exploration of the proposed parallelism capabilities.

#### 4. Appendix – Article Datasheets

---

# **ITAG PHASE 1 THRUST 1A TEST ARTICLE ANSWER KEY**

Information Sciences Institute  
University of Southern California

**FOR IRIS INTERNAL USE**

Modified: November 7, 2012

<b>0 Preface</b>	<b>4</b>
0.1 Overview	4
0.2 Errata List	4
<b>1 System Errata</b>	<b>5</b>
1.1 Errata	5
1.2 Features	5
1.3 Block Diagram	5
1.4 I/O Description	6
1.5 Memory Map	6
1.6 Resets	6
<b>2 ARM Subsystem Errata</b>	<b>7</b>
2.1 Errata	7
2.2 I/O Description	7
2.3 Technical Details	7
2.3.1 ARM Core	7
2.3.2 ARM Control Registers	7
2.3.3 GSM A5/1 Stream Cypher	8
<b>3 Thermal Classifier Subsystem Errata</b>	<b>10</b>
3.1 Errata	10
3.2 Block Diagram	10
3.3 I/O Description	10
3.4 Technical Details	10
3.4.1 Performance Monitors Infrastructure	10
<b>4 Memory Controller Subsystem Errata</b>	<b>14</b>
4.1 Errata	14
4.2 I/O Description	14
4.3 Technical Details	14
4.3.1 Passthrough Mode	14
<b>5 SPI Subsystem Errata</b>	<b>15</b>
5.1 Errata	15
<b>6 I2C Subsystem Errata</b>	<b>16</b>
6.1 Errata	16
<b>7 UART Subsystem Errata</b>	<b>17</b>
7.1 Errata	17
7.2 Features	17
7.3 Technical Details	17
<b>8 Sensor Subsystem Errata</b>	<b>19</b>
8.1 Overview	19
8.2 Features	19
8.3 Block Diagram	19
8.4 I/O Description	20
8.5 Technical Details	20
8.5.1 Control and Access	20
8.5.2 Address Map	20
8.5.3 Register Descriptions	21
8.5.4 Sensor Node Design and Operation	22
8.5.5 Programmable Ring Oscillator	23
8.5.6 NBTI Instrument and Measurements	24



**9 AXI4 Interconnect Errata 26**

9.1 Errata . . . . . 26

9.2 I/O Description . . . . . 26

9.3 Technical Details . . . . . 26

**10 Package Errata 27**

10.1 Errata . . . . . 27

10.2 Pad Frame . . . . . 27

# 0 | Preface

## 0.1 Overview

The ITAG Phase 1 Thrust 1A Test Article (TA1A) is a System-on-Chip (SoC) ASIC developed by USC Information Sciences Institute in support of the DARPA Integrity and Reliability of Integrated Circuits (IRIS) Thrust 1A.

This document describes differences between the delivered TA1A test article and the corresponding datasheet released to IRIS performers.

## 0.2 Errata List

Each difference between the TA1A test article and datasheet is listed below and numbered according to the ITAG internal tracking number.

- 106:** Unconnected Ring Oscillators. Two ring oscillators with no output were inserted into the test article. One of the two is always enabled, while the other is always disabled. This erratum is described in Section 1.1.
- 107:** Health Monitoring Sensors. An array of 16 ring oscillator sensors was inserted into the test article. This erratum is described in Chapter 8 and in sections 1.3, 1.4, 1.5, 1.6, and 10.2.
- 108:** GSM A5/1 Stream Cypher. A GSM A5/1 cypher core was attached to the ARM coprocessor. This erratum is described in Section 2.3.3.
- 109:** Performance Monitors. A collection of subsystem runtime performance monitors was inserted into the test article. This erratum is described in Chapter 3 (sections 3.2, 3.3, and 3.4.1) and in sections 2.2, 2.3.1, 9.2, and 9.3.
- 110:** I/O Pin for Memory Controller Passthrough Mode. An extra I/O pin was added to the test article to force the Memory Controller subsystem into passthrough mode. This erratum is described in sections 1.4, 4.2, 4.3.1, and 10.2.
- 111:** Minor Modifications to ARM. One extra instruction and two extra coprocessor registers were added to the ARM subsystem in the test article. This erratum is described in sections 2.3.1 and 2.3.2.
- 112:** Writable UART Counters. Writable UART counters were inserted into the test article to allow runtime baud rate adjustments. This erratum is described in Chapter 7 (sections 7.2 and 7.3) and in Section 1.2.
- 762:** Address Pin Count Mismatch. The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24. This erratum is described in sections 1.4 and 4.2.

# 1 | System Errata

## 1.1 Errata

Erratum 106: Two ring oscillators with no output were inserted into the test article. One of the two is always enabled, while the other is always disabled.

Erratum 107: An array of 16 ring oscillator sensors was inserted into the test article.

Erratum 110: An extra I/O pin was added was added to the test article to force the Memory Controller subsystem into passthrough mode.

Erratum 112: Writable UART counters were inserted into the test article to allow runtime baud rate adjustments.

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 1.2 Features

- Erratum 112: UART baud rates from 300 to 4,608,000

## 1.3 Block Diagram

Erratum 107: The Sensor subsystem is connected to the TA1A AXI4S Interconnect.

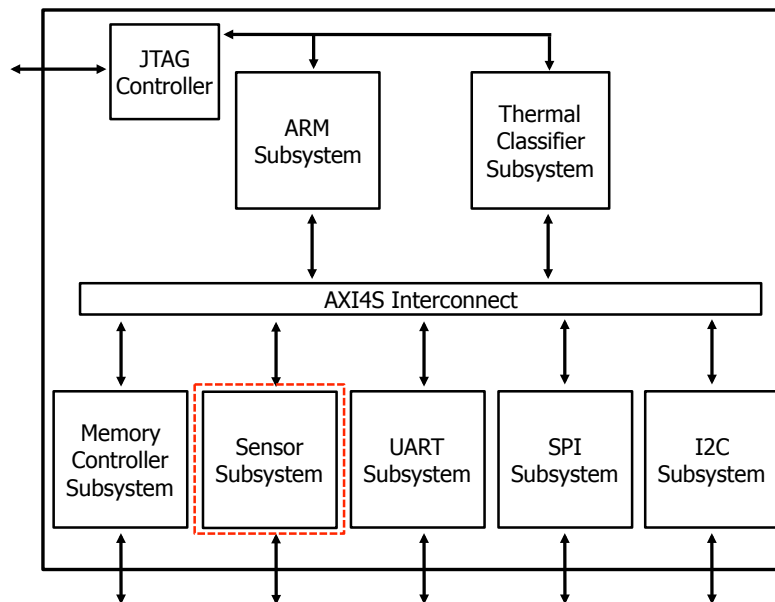


Figure 1.1: High-level block diagram of the TA1A System-on-Chip

## 1.4 I/O Description

Errata 107 and 110: The following I/O pins were added to the test article:

Table 1.1: Chip I/O Signals (Added)

Signal	In/Out	Width	Description
Sensor			
SENS_IN	In	1	Serial input for the sensor array
SENS_OUT	Out	1	Serial output from the sensor array
SENS_SCAN_EN	In	1	Scan enable
SENS_PWM	In	1	Pulse Width Modulation signal
Clock and Resets			
RESET_SENS_B	In	1	SENS subsystem reset (active low)

Erratum 762: The following I/O signal width was corrected:

Table 1.2: Chip I/O Signals (Corrected)

Signal	In/Out	Width	Description
Memory Controller			
MEM_ADDR	Out	28	Memory address

## 1.5 Memory Map

Erratum 107: The Sensor subsystem occupies the following space in the system memory map:

Table 1.3: TA1A System Memory Map

Address Range	Subsystem
0x00000000 – 0x0FFFFFFF	Memory Controller
0x10000000 – 0x1FFFFFFF	ARM
0x20000000 – 0x2FFFFFFF	Thermal Classifier
0x30000000 – 0x3FFFFFFF	Sensor
0x40000000 – 0x4FFFFFFF	SPI
0x50000000 – 0x5FFFFFFF	I2C
0x60000000 – 0x6FFFFFFF	UART
0x70000000 – 0xFFFFFFFF	[reserved]

## 1.6 Resets

Erratum 107: The subsystem I/O reset pins include the RESET\_SENS pin for the Sensor subsystem.

## 2 | ARM Subsystem Errata

### 2.1 Errata

Erratum 108: A GSM A5/1 cypher core was attached to the ARM coprocessor.

Erratum 109: A collection of subsystem run-time performance monitors was inserted into the test article.

Erratum 111: One extra instruction and two extra coprocessor registers were added to the ARM subsystem in the test article.

### 2.2 I/O Description

Erratum 109: The following I/O pins were added to the ARM Subsystem.

Table 2.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	Out	1	ARM processor stall signal

### 2.3 Technical Details

#### 2.3.1 ARM Core

Erratum 109: The ARM processor's fetch stall signal, known as arm\_cpuwait, is connected from the ARM subsystem to the SVD Subsystem. The processor stalls when the processor performs I/O transactions to memory. More details regarding the monitoring of the ARM processor's cpuwait signal can be found in Chapter 3.

Erratum 111: A new bounded multiply operation MULB has been added to the ARM instruction set. The regular MUL instruction treats the <Rd> opcode bits [15:12] as reserved, and requires that they be set to zero. When <Rd> is non-zero, the processor instead executes the MULB instruction, and uses Rd as a bound on the result. If the product exceeds the bound, the bound is returned instead of the product. In all other respects the MUL and MULB instructions are identical, and MULB reduces to MUL when <Rd> is zero.

**MULBcdS** regD, RegA, RegB, RegC

Multiply RegA and RegB, bounded by RegC, and place into RegD. If RegC is r0, no bound is used, and the operation is MULcdS.

$$\text{RegD} = (\text{RegA} \times \text{RegB}) > \text{RegC} ? \text{RegC} : (\text{RegA} \times \text{RegB})$$

Execute only if cd is true.

Set flags if S is specified.

#### 2.3.2 ARM Control Registers

The ARM VL86C020 and derivative processors include control registers used for cache control and device identification. These control registers are accessible as built-in Coprocessor 15.

Erratum 111: Coprocessor 15 Control Register 0 (Identity Register) can be written through the JTAG register DATA\_OUT, and read by the ARM. This allows the user to override the standard ARM v2 processor identification code.

Erratum 111: Coprocessor 15 Control Register 1 (Cache Flush) is now an actual 32-bit register that can be written by the ARM, and read through the JTAG register DATA\_IN. This provides a debug mechanism, allowing the user to share data on the JTAG port. Writing to Coprocessor 15 Control Register 1 still forces a cache flush as expected.

### 2.3.3 GSM A5/1 Stream Cypher

Erratum 108: This entire subsection has been added as an erratum.

A GSM A5/1 stream cypher core is attached to the ARM core through Coprocessor 15. This core is used to create a keystream that can be used to encrypt plain text. The cypher core implements GSM A5/1 to produce a running keystream by XORing the most significant bits of 3 Linear Feedback Shift Registers (LFSRs). The core can reset its contents and then accept a 64-bit externally supplied secret session key and a 22-bit frame number to prepare for keystream generation. During the preparation process, the least significant bit of each LFSR is XORed with a corresponding bit from the secret session key, and after that with a corresponding bit from the frame number. During this preparation phase, all LFSRs operate continuously with regular clocking. The eight possible modes of the 3-bit address port can be used for the purpose of loading the secret session key and frame number.

Once the secret session key and frame number have been loaded into the LFSRs, the address lines can be used to place the core in keystream generation mode to produce a pair of 114-bit keystreams. These keystreams are grouped into 32-bit words, and accessed by the ARM core through the Coprocessor 15 interface.

During the A5/1 keystream generation phase the core uses a combination of the three LFSRs operated in an irregular clocking scheme to iteratively generate 3 separate sequences of bits, which are then XORed to generate a bit of keystream per clock cycle. The A5/1 LFSR parameters are shown in Table 2.2. LFSRs whose clocking bit equals the majority value of all clocking bits will shift their contents. If any of the LFSRs does not match the majority value, it is stalled until its clock bit equals the majority value.

Table 2.2: GSM A5/1 Parameters

LFSR	Length	Feedback Polynomial	Clocking Bit
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8
2	22	$x^{22} + x^{21} + 1$	10
3	23	$x^{23} + x^{22} + x^{21} + x^8 + 1$	10

The A5/1 algorithm requires three LFSRs of bit lengths 19, 22, and 23, but the design implements them using three 32 bit registers, with the lengths of the LFSRs being initialized prior to keystream generation. Consequently, each bit holding and bit manipulating function associated with each bit position in the LFSRs is designed as a generic unit-block circuit. Through the use of several control signals, a unit-block can operate in regular or irregular clocking modes and can appropriately XOR its contents with a value received from polynomial evaluation performed on more significant bits. This means that the core can also be used as a pseudo-random number generator, by initializing the LFSR lengths, polynomials, and clocking bits.

The core is connected to the ARM core via a 32-bit coprocessor interface. It is the responsibility of the software on the ARM core to appropriately load and use the two 114-bit keystream pairs. In addition, the module has a 3-bit address port and a read/write strobe signal interface with the coprocessor. Once a keystream has been generated, the plaintext encryption can be done outside the core.

The A5/1 core is initialized by writing to Coprocessor 15 register CR6. Keystream data is obtained from the core by reading from Coprocessor 15 register CR8. These registers use self-incrementing counters, so data must always be written to or read from them in groups of eight words. The initialization data sequence is presented in Table 2.3, and the keystream data sequence is presented in Table 2.4.

Table 2.3: Initialization Sequence: Coprocessor 15 Register CR6

Index	Bits	Description
0	[7:0]	LFSR 0 length
0	[15:8]	LFSR 1 length
0	[23:16]	LFSR 2 length
0	[31:24]	Reserved
1	[31:0]	LFSR 0 polynomial
2	[31:0]	LFSR 1 polynomial
3	[31:0]	LFSR 2 polynomial
4	[3:0]	LFSR 0 clocking bit
4	[7:4]	LFSR 1 clocking bit
4	[11:8]	LFSR 2 clocking bit
4	[31:12]	Reserved
5	[31:0]	LFSR 0 session key
6	[31:0]	LFSR 1 session key
7	[21:0]	LFSR 2 session key

Table 2.4: Keystream Sequence: Coprocessor 15 Register CR8

Index	Description
0	Keystream 0 bits [31:0]
1	Keystream 0 bits [63:32]
2	Keystream 0 bits [95:64]
3	Keystream 0 bits [127:96]
4	Keystream 1 bits [31:0]
5	Keystream 1 bits [63:32]
6	Keystream 1 bits [95:64]
7	Keystream 1 bits [127:96]



### 3 | Thermal Classifier Subsystem Errata

#### 3.1 Errata

Erratum 109 Performance Monitors. A collection of subsystem run-time performance monitors was inserted into the test article.

#### 3.2 Block Diagram

Erratum 109: The following block diagram reflects the modifications made to the Thermal Classifier Subsystem.

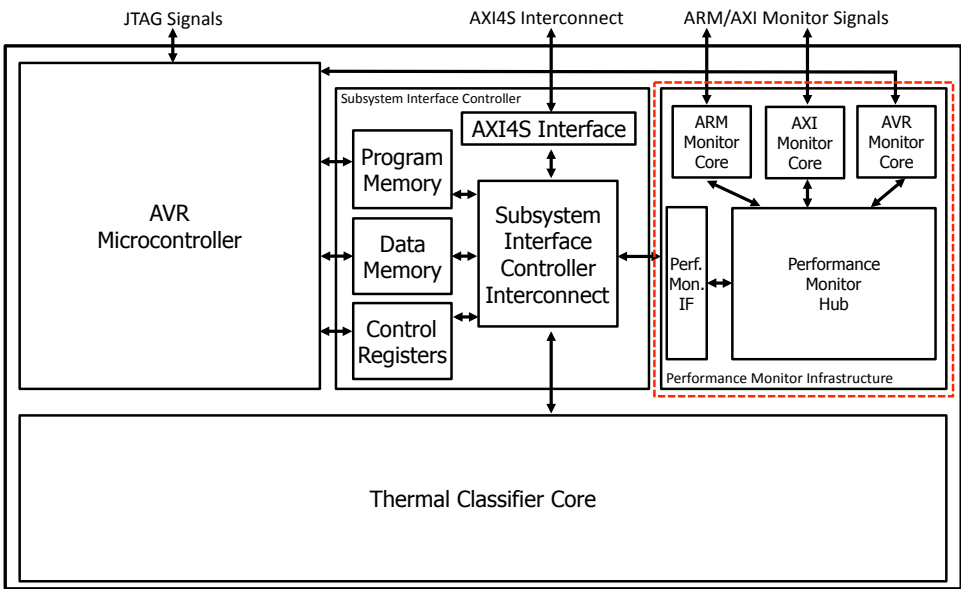


Figure 3.1: Thermal Classifier Subsystem Block Diagram

#### 3.3 I/O Description

Erratum 109: The following I/O pins were added to the Thermal Classifier Subsystem.

Table 3.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	In	1	ARM processor stall signal
axi_ports_empty	In	7	AXI Interconnect Input FIFO empty signal
axi_ports_full	In	7	AXI Interconnect Input FIFO full signal
axi_ports_valid	In	7	AXI Interconnect Input FIFO valid signal
axi_ports_ready	In	7	AXI Interconnect Input FIFO ready signal

#### 3.4 Technical Details

##### 3.4.1 Performance Monitors Infrastructure

Erratum 109: This entire subsection has been added as an erratum.

The performance monitor infrastructure provides run-time system information. The information can be collected and used by a designer to better understand the system performance under various loads and conditions. The system uses individual cores to monitor the ARM processor, AVR processor, and AXI4S interconnect. A designer can enable or disable monitoring and capture or reset each monitor core's data. The monitoring infrastructure is composed of the following blocks:

- Performance Monitor Interface
- Performance Monitor Hub
- ARM Performance Monitor Core
- AVR Performance Monitor Core
- AXI4S Interconnect Monitor Core

The ARM subsystem and the AXI4S interconnect are separate from the Thermal Classifier subsystem, but their monitoring cores reside within the Thermal Classifier subsystem. Figure 3.1 shows the performance monitor infrastructure integrated into the Thermal Classifier subsystem, including the subsystem I/O ports added for external monitoring of the ARM subsystem and AXI4S interconnect.

#### 3.4.1.1 Performance Monitor Interface

The system interacts with the Performance Monitor through the Performance Monitor Interface. An additional port was added to Subsystem Interface Controller (SIC) interconnect. This port connects to the Performance Monitor Interface at address 0x25000000. The interface also adds separate 16-element deep FIFOs on the transmit and receive ports to buffer commands and data going to and from the system.

#### 3.4.1.2 Performance Monitor Hub

The Performance Monitor Hub aggregates commands from the system and passes them on to the specified performance monitor core. Table 3.2 defines the supported commands.

Table 3.2: Performance Monitor Commands

Command	Description
0x0	Retrieve all data from all performance monitors
0x1	Retrieve all data from a specific performance monitor
0x2	Retrieve a specific data word from all performance monitors
0x3	Retrieve a specific data word from one performance monitor
0x4	Reset data for all performance monitors
0x5	Reset data for a specific performance monitor
0x6	Enable data collection for all performance monitors
0x7	Enable data collection for a specific performance monitor
0x8	Disable data collection for all performance monitors
0x9	Disable data collection for a specific performance monitor

Table 3.3 enumerates the performance monitor cores. These numbers can be combined with commands to designate a specific performance monitor.

Table 3.3: Performance Monitor Cores Numeric Representation

Number	Core Name
0	AVR Processor Performance Monitor
1	ARM Processor Performance Monitor
2	AXI Interconnect Performance Monitor

Table 3.4 describes the Performance Monitor Hub Command Register at address 0x25000000.

Table 3.4: Performance Monitor Hub Command Register

Bit number	Access	Description
[31:12]	—	Reserved
[11:8]	w	Monitor number (Table 3.3)
[7:4]	—	Reserved
[3:0]	w	Command (Table 3.2)

After a command is issued, the resulting data can be read from address 0x25000000. The data returned depends on the command that was issued. The first word of data indicates how many monitors are included in the results. Then for each monitor, the number of data words, followed by the actual data words are returned. A simple C program with a double-nested loop can be used to iterate over each monitor and then over each datum.

### 3.4.1.3 ARM Performance Monitor Core

The ARM performance monitor core receives input signal `arm_cpawait`. When the `arm_cpawait` signal is high, the ARM processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `arm_cpawait` signal is active. Combined with the total run-time of the ARM subsystem, a user can quickly understand the utilization of the processor core. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ARM monitor includes two 64-bit timers. Timer 0 measures the idle time when `arm_cpawait` is asserted. Timer 1 measures the active run time, when `arm_cpawait` is not asserted. The monitor data is described in Table 3.5.

Table 3.5: ARM Performance Monitor Core's Data Order

Word	Description
0	Timer 0: ARM processor idle timer [31:0] data
1	Timer 0: ARM processor idle timer [63:32] data
2	Timer 1: ARM processor run timer [31:0] data
3	Timer 1: ARM processor run timer [63:32] data

### 3.4.1.4 AXI4S Interconnect Performance Monitor Core

The AXI4S interconnect performance monitor core receives inputs `axi_port_empty`, `axi_port_full`, `axi_port_valid`, and `axi_port_ready`. These signals reflect the AXI4S interconnect input FIFO status. The ports in the TA1A interconnect each have FIFOs to buffer incoming data. The FIFO status is useful for understanding the utilization of the interconnect and the load distribution of an application on the system. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AXI4S monitor data includes one 32-bit word indicating the status of the input FIFOs. The AXI4S interconnect has 7 ports. The status information is divided into four groups as shown in Table 3.6. Within each group the bit position corresponds to the port number.

Table 3.6: AXI4S Interconnect Status Register

Bit number	Access	Description
[31:28]	—	Reserved
[27:21]	r	AXI4S input FIFO ready signals
[20:14]	r	AXI4S input FIFO valid signals

Table 3.6: AXI4S Interconnect Status Register

Bit number	Access	Description
[13:7]	r	AXI4S input FIFO full signals
[6:0]	r	AXI4S input FIFO empty signals

### 3.4.1.5 AVR Performance Monitor Core

The AVR performance monitor core receives inputs `avr_cpuwait`, `avr_pc`, and `avr_inst`. When the `avr_cpuwait` signal is high, the AVR processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `avr_cpuwait` signal is active. Combined with the total run-time of the AVR subsystem, a user can quickly understand the utilization of the processor core. The monitor can also capture the current value of the program counter and the current instruction that is being executed. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AVR monitor includes two 64-bit timers and two 32-bit words for the program counter and current instruction. Timer 0 measures the idle time when the `avr_cpuwait` signal is asserted. Timer 1 measures the active run time, when the `avr_cpuwait` signal is not asserted. The monitor data is described in Table 3.7.

Table 3.7: AVR Performance Monitor Core's Data Order

Word	Description
0	Timer 0: AVR processor idle timer [31:0] data
1	Timer 0: AVR processor idle timer [63:32] data
2	Timer 1: AVR processor run timer [31:0] data
3	Timer 1: AVR processor run timer [63:32] data
4	AVR processor program counter
5	AVR processor instruction register

# 4 | Memory Controller Subsystem Errata

## 4.1 Errata

Erratum 110: An extra I/O pin was added to force the Memory Controller subsystem into passthrough mode.

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 4.2 I/O Description

Erratum 110: The following I/O pins was added to the test article:

Table 4.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
MEM_PASS_MODE	In	1	Force Memory subsystem into passthrough mode

Erratum 762: The following I/O signal width was corrected:

Table 4.2: Subsystem I/O Signals (Changed)

Signal	In/Out	Width	Description
MEM_ADDR	Out	28	Off-chip memory address. Provides the base address (or the start address in case of a burst) of the data to be accessed.

## 4.3 Technical Details

### 4.3.1 Passthrough Mode

Erratum 110: The Memory Controller subsystem can be forced into Passthrough mode by driving the device I/O MEM\_PASS\_MODE pin high. The documented method of entering Passthrough mode by asserting the ACK signal and holding the two MSBs of MEM\_DATA\_IN high also remains valid.

## **5 | SPI Subsystem Errata**

### **5.1 Errata**

No errata exist for this subsystem.

## 6 | I2C Subsystem Errata

### 6.1 Errata

No errata exist for this subsystem.

# 7 | UART Subsystem Errata

## 7.1 Errata

Erratum 112: Writable UART counters were inserted into the test article to allow runtime baud rate adjustments.

## 7.2 Features

- Erratum 112: Baud rates from 300 to 4,608,000

## 7.3 Technical Details

Erratum 112: The UART supports operations to receive and transmit data, to get or set the baud rate, to get the FIFO status, and to acquire, check, or release a mutex. The operation requested is determined by the read or write address from Table 7.1.

Table 7.1: UART Address Summary

Address	Description
0x60000000	Normal Operation
0x60000004	Get/Set Baud Low
0x60000008	Get/Set Baud High
0x6000000C	Get FIFO Status
0x60000010	Check Mutex
0x60000110	Acquire Mutex
0x60000210	Release Mutex

Erratum 112: The UART baud rate is controlled by two 32-bit registers. The low 12 bits at address 0x60000004 set the baud frequency and the low 16 bits at address 0x60000008 set the baud limit. These registers together set two internal counters that configure the baud clock.



Erratum 112: The UART default baud rate is 115,200 bps. Table 7.2 shows the baud rate settings to use if the system clock frequency is 100 MHz.

Table 7.2: UART Settings

Baud Rate	baud_freq	baud_limit
300	0x0003	0xF421
600	0x0003	0x7A0F
1,200	0x0003	0x3D06
2,400	0x0006	0x3D03
4,800	0x000C	0x3CFD
9,600	0x0018	0x3CF1
14,400	0x0024	0x3CE5
19,200	0x0030	0x3CD9
28,800	0x0048	0x3CC1
38,400	0x0060	0x3CA9
56,000	0x001C	0x0C19
57,600	0x0090	0x3C79
<b>115,200<sup>†</sup></b>	<b>0x0120</b>	<b>0x3BE9</b>
128,000	0x0040	0x0BF5
153,600	0x0180	0x3B89
230,400	0x0240	0x3AC9
256,000	0x0080	0x0BB5
460,800	0x0480	0x3889
921,600	0x0900	0x3409
1,382,400	0x0D80	0x2F89
2,304,000	0x0480	0x07B5
4,608,000	0x0900	0x0335

<sup>†</sup> Default baud rate

Erratum 112: The baud settings in Table 7.2 can be calculated from the desired baud rate as follows:

$$Baud\_freq = \frac{16 \times baud\_rate}{gcd(system\_clock\_freq, 16 \times baud\_rate)}$$

$$Baud\_limit = \frac{system\_clock\_freq}{gcd(system\_clock\_freq, 16 \times baud\_rate)} - baud\_freq$$

## 8 | Sensor Subsystem Errata

### 8.1 Overview

Erratum 107: This entire chapter has been added as an erratum.

The ITAG sensor array consists of 16 sensor nodes connected in a daisy chain, control logic, and independent off-chip and on-chip interfaces. Each sensor node contains a programmable ring oscillator and a frequency counter. The ring oscillators can be sampled in various configurations and operating conditions, allowing the inference of several physical parameters. A Pulse Width Modulation (PWM) signal is used to activate/de-activate any or all of the ring oscillators. The PWM signal can be driven by an external pin (by default), or via software writes to a control register. The daisy chain acts as both a scan chain and a conduit for any ring oscillator output to be forwarded downstream and off-chip.

### 8.2 Features

- Allows measurement of delays at 16 locations across the chip
- Allows measurement of Negative Bias Temperature Instability (NBTI) via specialized circuit
- Control and access from either on chip or off chip
- Sample rate up to 2.5 million samples per second
- Can be operated concurrently with system operation or while the system is held in reset
- Ability to drive any of the 16 ring oscillator signals off chip

### 8.3 Block Diagram

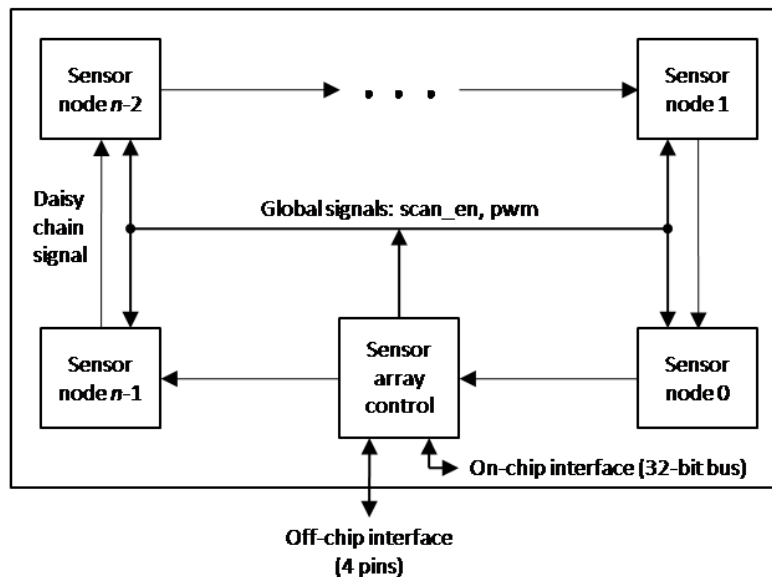


Figure 8.1: Block diagram

## 8.4 I/O Description

Table 8.1: Subsystem I/O Signals

Signal	In/Out	Width	Description
SENS_IN	In	1	Off-chip serial input for the sensor array scan chain. Valid when SENS_SCAN_EN is asserted. Setup and hold times are with respect to the system clock. When controlling the sensor array with internal signals rather than off-chip inputs, this must be tied to 1. The length of the scan chain depends on the number of sensor nodes in Bypass Mode. The maximum length is 16 nodes * 16b = 256b.
SENS_OUT	Out	1	Sensor array serial output. During scan, this acts as the scan output. Data is scanned out at the system clock rate. The amount of data available to be scanned out depends on the number of sensor nodes that are bypassed; the maximum length of the scan chain is 16 nodes * 16b = 256b. Output data is valid whenever a scan is performed, whether controlled by the SENS_SCAN_EN input or the internal scan enable signal. When scan is not underway, this output by default reflects the input to the sensor array (either the value of SENS_IN or the value of the least significant bit of the control register). The output can optionally be used to observe any ring oscillator output, by setting the appropriate bits in the sensor nodes.
SENS_SCAN_EN	In	1	Off-chip scan enable signal. When controlling the sensor array with internal signals rather than off-chip inputs, this must be tied to 0.
SENS_PWM	In	1	Pulse Width Modulation signal which gates ring oscillators on/off. Can be asserted and deasserted asynchronously. Has an effect whenever the off-chip inputs are enabled; has no effect otherwise.
RESET_SENS_B	In	1	External reset pin for the sensor subsystem. Active low. Can be used to hold the subsystem in reset even when the chip reset pin (RESET_B) is deasserted.

## 8.5 Technical Details

### 8.5.1 Control and Access

The sensor array can be controlled via an off-chip interface by manipulating three chip input pins. Alternatively, it can be controlled by writing to a control register from one of the on-chip processors. Only one of the two interfaces can be actively controlling the sensor array at one time; the selected interface is determined by a control register bit. When the chip is reset, the default is to use the off-chip interface.

Sensor array data can be observed at either of the two interfaces, even though the array is controlled via a single interface. Serial data can be driven out through an output pin on the off-chip interface, and a parallel data can be read from a register via the on-chip interface.

### 8.5.2 Address Map

Table 8.2: Sensor Array Address Map

Address	Description
0x30000000	Sensor Array Control Register
0x30000004	Sensor Array Timer Register

### 8.5.3 Register Descriptions

Note: writes to the registers only have an effect when SENS\_IN = 1 and SENS\_SCAN\_EN = 0.

Table 8.3: Sensor Array Control Register

Bit number	Access	Description
[31:24]	n/a	Reserved
[23:19]	r	Scan count. Automatically set to 01111b when the Advance Scan Chain bit is written with a 1; decrements as the scan chain is advanced.
18	n/a	Reserved
17	r/w	Interface Select. 0: off-chip inputs are enabled (IN, SCAN_EN, PWM); 1: on-chip signals are enabled (control register scan data, internal scan enable, timer-generated PWM)
16	r/w	Advance Scan Chain.
15	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Bypass Scan Chain. 1 = only a single bit (bit 15) of the node's 16-bit register will be included in the next scan operation; 0 = all 16 bits of the node's register will be included in the next scan operation. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
[14:12]	r/w	Scan data. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data. This field is a don't-care when writing configuration data to be scanned in to a sensor node.
11	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Enable 16-Inverter Chain. 1 = the optional 16-inverter chain is included in the ring oscillator path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
10	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Enable 8-Inverter Chain. 1 = the optional 8-inverter chain is included in the ring oscillator path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
9	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Enable 4-Inverter Chain. 1 = the optional 4-inverter chain is included in the ring oscillator path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
8	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Enable 2-Inverter Chain. 1 = the optional 2-inverter chain is included in the ring oscillator path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
7	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents NBTI Chain 1 Bias Value. 1 = unstressed state; 0 = stressed state. This bit must be set to 0 temporarily in order to sample the ring oscillator with NBTI Chain 1 in the path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
6	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents NBTI Chain 2 Bias Value. 1 = unstressed state; 0 = stressed state. This bit must be set to 0 temporarily in order to sample the ring oscillator with NBTI Chain 2 in the path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
5	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Measure NBTI Chain 1. 1 = the optional NBTI chain 1 is included in the ring oscillator path. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.

Table 8.3: Sensor Array Control Register

Bit number	Access	Description
4	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Measure NBTI Chain 2. 1 = the optional NBTI chain 2 is included in the ring oscillator path (requires that Measure NBTI Chain 1 be deasserted). When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
3	r/w	Scan data. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data. This field is a don't-care when writing configuration data to be scanned in to a sensor node.
2	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Ring Oscillator Enable (active low). 1 = ring oscillator stays off during PWM assertions; 0 = ring oscillator turns on during PWM assertions. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
1	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents Select Clock from Upstream. 1 = the daisy chain input is used as a clock for the node's counter; 0 = the local ring oscillator is used as the clock for the node's counter. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.
0	r/w	Scan data. When used as configuration data to be scanned in to a sensor node, this bit represents the Output Mode Select. 1 = the node's ring oscillator signal is propagated; 0 = the node's daisy chain input is propagated. When reading result data after advancing the scan chain, this is part of the 16 bits of scan data.

Note: after advancing the scan chain by one sensor node position, bits [15:0] typically represent the 16-bit ring oscillator count from one sensor node. An exceptional case is when one or more nodes has been bypassed from the scan chain; in that case some nodes will only have 1 bit in the scan data, and thus ring oscillator counts will not always line up with the 16-bit field in the register.

Table 8.4: Sensor Array Timer Register

Bit number	Access	Description
[31:0]	r/w	Timer value. Represents the current value of the decrementing timer. A non-zero timer value causes the internal PWM signal to be asserted as long as the timer value is non-zero. The internal PWM signal is used by the sensors, provided the sensor array is under control of the internal signals (as dictated by the Interface Select bit in the Control Register). The timer value is set by a write to this register; subsequently the value automatically decrements each clock cycle if greater than 0. The value stops at 0 and the internal PWM signal deasserts.

### 8.5.4 Sensor Node Design and Operation

The basic sequence of operation involves scanning in a configuration for the sensor array (e.g., specifying which ring oscillators to sample and in which modes), sampling the frequency of one or more ring oscillators over a deterministic period, and then scanning out the data. Scanning out data and scanning in the next configuration can be performed simultaneously.

When using the off-chip interface, data is scanned in through the SENS\_IN pin. When using the on-chip interface, data is scanned by writing a configuration value for a single sensor node at a time. Scanning the entire N-node

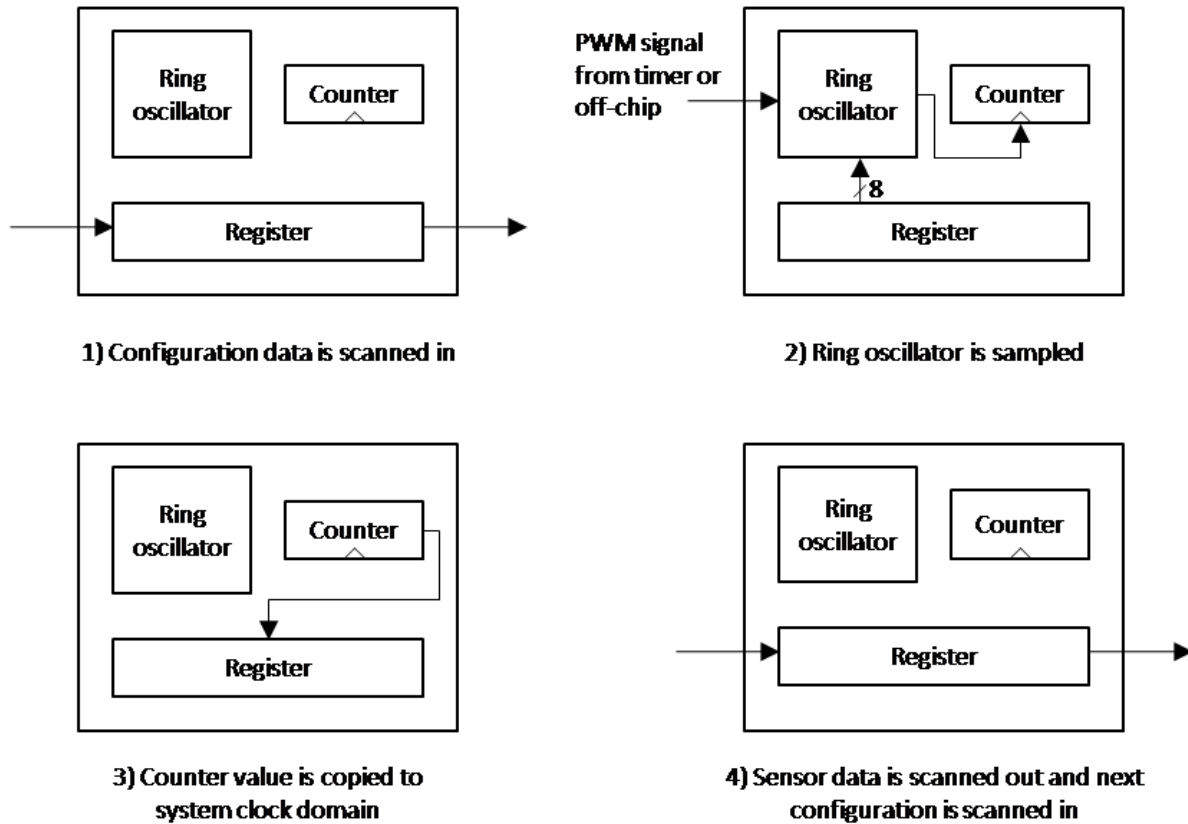


Figure 8.2: Operational concept of the sensor node

array requires  $N$  register writes. After each write, the sensor array control logic advances the daisy chain by 1 word.

The sensor array is accessed by scanning the daisy chain at the system clock frequency. To improve the sample rates, the logic design allows individual sensor nodes to be bypassed when scanning. In bypass mode there is just one cycle of delay through the node. The minimum time to scan in a new configuration and simultaneously scan out the previous result is  $t_{min\_scan} = (N - 1 + M) t_{clk}$ , where  $N - 1$  sensor nodes are bypassed,  $M$  is the width of the counter in the activated node, and  $t_{clk}$  is the system clock period. As an example, with 16 nodes, 16-bit counters, and a 10 ns period, the scan overhead is  $t_{min\_scan} = (16 - 1 + 16)(10 \text{ ns}) = 310 \text{ ns}$ .

The total time required for a sample is the time to scan plus the PWM period plus a small amount of dead time in between each step (e.g. to allow time for the PWM signal to be synchronized to the ring oscillator clock). The minimum total time is approximately 400 ns, assuming an extremely narrow PWM period. This corresponds to a maximum rate of 2.5M samples/s.

### 8.5.5 Programmable Ring Oscillator

The basic concept for the programmable ring oscillator is shown in Figure 8.3. It includes a programmable inverter chain and a negative-bias temperature instability (NBTI) instrument. The oscillator can be configured as desired and then activated for the desired sample period.

The inverter chain can be configured for any even number of stages between 0 and 30:

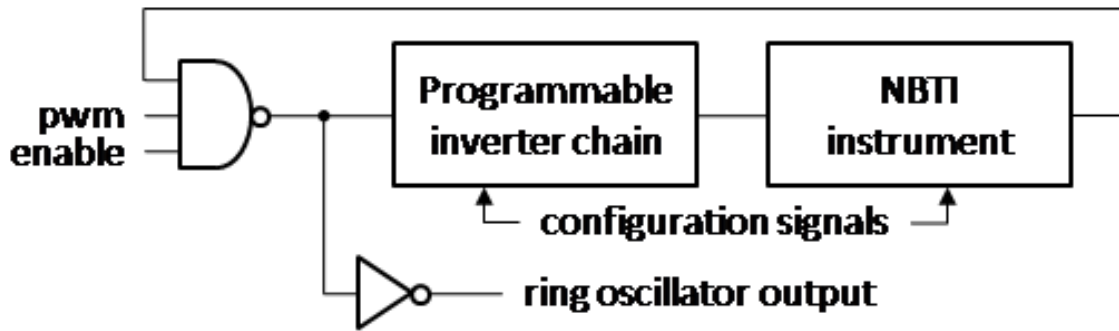


Figure 8.3: Programmable ring oscillator

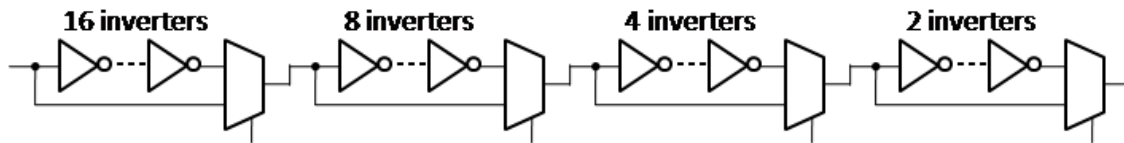


Figure 8.4: Selectable number of inverter stages

### 8.5.6 NBTI Instrument and Measurements

The NBTI instrument consists of two chains of gates which can be independently biased, allowing differential measurements of NBTI degradation. The chains consist of minimum-sized OR-AND-INVERT cells (oai21\_1x); this type of cell allows one PMOS device per cell to be fully controlled while a string of cells are chained together. For a competing method, see “Ring oscillator circuit structures for measurement of isolated NBTI/PBTI effects,” Kim et al., IEEE International Conference on Integrated Circuit Design and Technology, 2008. The method by Kim et al. uses NOR gates but does not provide full control; the topology allows NBTI effects to be separated from PBTI effects, but causes half of the PMOS devices under test to be negatively biased even when the circuit is in the least stressed state. Our design allows the DUTs to be configured for all stress, no stress, or measurement mode. Transistor-level views of an OAI gate are shown in Figure 8.5, showing the configurations used to stress, unstress, and measure the PMOS transistor under test.

A gate-level view of the instrument is shown in Figure 8.6. This example shows just four oai21\_1x gates per chain; the actual number is 10.

In normal mode, the chains are bypassed from the ring oscillator path and are held in either a stressed or unstressed state. During this static bias, the ring oscillator can still be used without the NBTI instrument (note in Figure 8.6 that the “from oscillator path” can be driven directly to the output through a mux). The wearout can be accelerated by externally controlling the core voltage and/or the temperature. In measurement mode, one of the chains is inserted into the ring oscillator path so that wearout can be inferred via the ring oscillator frequency. During measurement mode, half of the oscillator pulses will traverse the PMOSes in the odd-numbered gates, and the other half will traverse the PMOSes in the even-numbered gates. To help isolate the effect, the remainder of the ring oscillator can be configured to be very short (e.g. 2 inverters instead of 30), so that the chain makes up a significant portion of the overall ring delay.

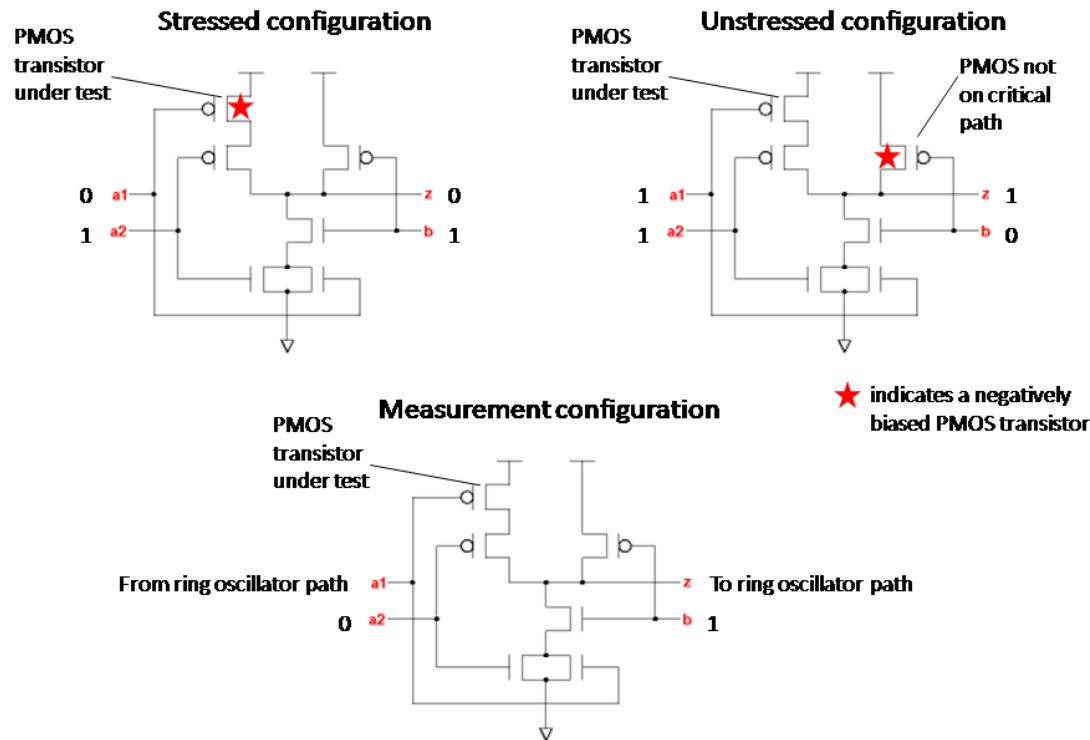


Figure 8.5: Stressed configuration (upper left); unstressed configuration (upper right); measurement configuration (lower center)

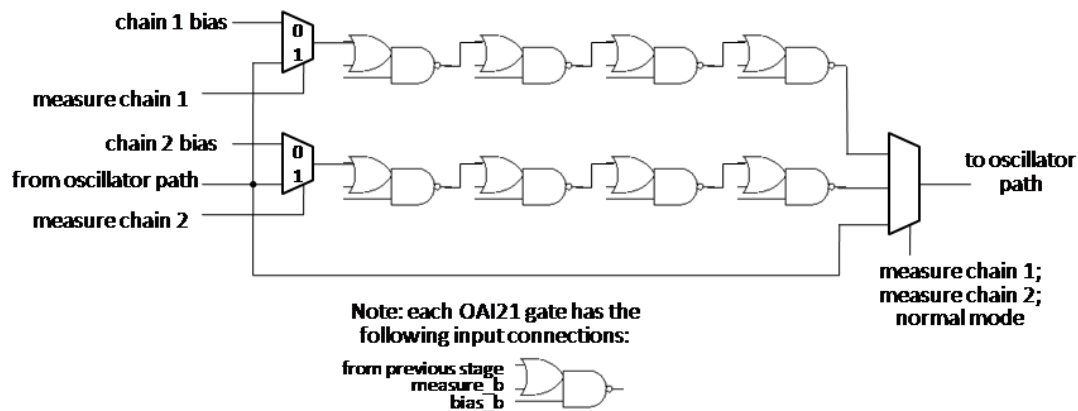


Figure 8.6: NBTI instrument



# 9 | AXI4 Interconnect Errata

## 9.1 Errata

Erratum 109: A collection of subsystem run-time performance monitors was inserted into the test article.

## 9.2 I/O Description

Erratum 109: The following I/O pins were added to the AXI4 Interconnect.

Table 9.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
axi_ports_empty	Out	7	AXI Interconnect Input FIFO empty signal
axi_ports_full	Out	7	AXI Interconnect Input FIFO full signal
axi_ports_valid	Out	7	AXI Interconnect Input FIFO valid signal
axi_ports_ready	Out	7	AXI Interconnect Input FIFO ready signal

## 9.3 Technical Details

Erratum 109: Each input port's FIFO status signals in the AXI interconnect are routed to the Thermal Classifier Subsystem. More details regarding the monitor of the FIFO signals can be found in Chapter 3.

## 10 | Package Errata

### 10.1 Errata

Erratum 110: An extra I/O pin was added to force the Memory Controller subsystem into passthrough mode.

Erratum 107: An array of 16 ring oscillator sensors was inserted into the test article.

### 10.2 Pad Frame

Errata 107 and 110: Six I/O pins were added in support of the Sensor subsystem and the Memory Controller passthrough mode.

Table 10.1: TA1A Pad Frame

Signal	Edge	CCW	Pad
SENS_SCAN_EN	W	66	M10
SENS_PWM	W	71	N9
SENS_OUT	W	72	N10
SENS_IN	W	73	N11
RESET_SENS_B	W	76	P8
MEM_PASS_MODE	S	135	A15

---

# **ITAG PHASE 1 THRUST 1B TEST ARTICLE ANSWER KEY**

Information Sciences Institute  
University of Southern California

**FOR IRIS INTERNAL USE**

Modified: November 7, 2012

<b>0 Preface</b>	<b>3</b>
0.1 Overview . . . . .	3
0.2 Errata List . . . . .	3
<b>1 System Errata</b>	<b>4</b>
1.1 Errata . . . . .	4
1.2 I/O Description . . . . .	4
<b>2 ARM Subsystem Errata</b>	<b>5</b>
2.1 Errata . . . . .	5
2.2 I/O Description . . . . .	5
2.3 Technical Details . . . . .	5
2.3.1 ARM Core . . . . .	5
2.3.2 ARM Control Registers . . . . .	5
2.3.3 Wishbone Debug Interface . . . . .	6
2.3.4 GSM A5/1 Stream Cypher . . . . .	6
<b>3 SVD Subsystem Errata</b>	<b>8</b>
3.1 Errata . . . . .	8
3.2 Block Diagram . . . . .	8
3.3 I/O Description . . . . .	8
3.4 Technical Details . . . . .	9
3.4.1 SVD Reordering . . . . .	9
3.4.2 Performance Monitors Infrastructure . . . . .	9
<b>4 Memory Controller Subsystem Errata</b>	<b>12</b>
4.1 Errata . . . . .	12
4.2 I/O Description . . . . .	12
4.3 Technical Details . . . . .	12
4.3.1 Passthrough Mode . . . . .	12
<b>5 SPI Subsystem Errata</b>	<b>13</b>
5.1 Errata . . . . .	13
<b>6 I2C Subsystem Errata</b>	<b>14</b>
6.1 Errata . . . . .	14
<b>7 UART Subsystem Errata</b>	<b>15</b>
7.1 Errata . . . . .	15
<b>8 VGA Subsystem Errata</b>	<b>16</b>
8.1 Errata . . . . .	16
8.2 I/O Description . . . . .	16
8.3 Technical Details . . . . .	16
<b>9 AXI4 Interconnect Errata</b>	<b>17</b>
9.1 Errata . . . . .	17
9.2 I/O Description . . . . .	17
9.3 Technical Details . . . . .	17
9.3.1 Crossbar Switch . . . . .	17
9.3.2 Arbitration . . . . .	17

# 0 | Preface

## 0.1 Overview

The ITAG Phase 1 Thrust 1B Test Article (TA1B) is a System-on-Chip (SoC) netlist developed by USC Information Sciences Institute in support of the DARPA Integrity and Reliability of Integrated Circuits (IRIS) Thrust 1B.

This document describes differences between the delivered TA1B test article and the corresponding datasheet released to IRIS performers.

## 0.2 Errata List

Each difference between the TA1B test article and datasheet is listed below and numbered according to the ITAG internal tracking number.

- 548:** Modification to Interconnect Port Scheduling. The AXI4S interconnect uses Round Robin arbitration with priority given to higher port numbers. This erratum is described in Section 9.3.2.
- 549:** Expanded ARM JTAG Capability. The JTAG interface can be used to read and write the ARM program counter. This erratum is described in Section 2.3.3.
- 550:** GSM A5/1 Stream Cypher. A GSM A5/1 cypher core was attached to the ARM coprocessor. This erratum is described in Section 2.3.4.
- 551:** Performance Monitors. A collection of subsystem runtime performance monitors was inserted into the test article. This erratum is described in Chapter 3 (sections 3.2, 3.3, and 3.4.2), and sections 2.2, 2.3.1, 9.2, and 9.3.1.
- 552:** I/O Pin for VGA Resolution. An extra I/O pin was added to the test article to force the VGA subsystem into high-resolution mode. This erratum is described in sections 1.2, 8.2, and 8.3.
- 553:** I/O Pin for SVD Result Order. An extra I/O pin was added to the test article to change the order of the SVD subsystem results. This erratum is described in sections 1.2, 3.3, and 3.4.1.
- 554:** I/O Pin for Memory Controller Passthrough Mode. An extra I/O pin was added to the test article to force the Memory Controller subsystem into passthrough mode. This erratum is described in sections 1.2, 4.2, and 4.3.1.
- 555:** Minor Modifications to ARM. One extra instruction and two extra coprocessor registers were added to the ARM subsystem in the test article. This erratum is described in sections 2.3.1 and 2.3.2.
- 762:** Address Pin Count Mismatch. The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24. This erratum is described in sections 1.2 and 4.2.

# 1 | System Errata

## 1.1 Errata

Erratum 549: The JTAG interface can be used to read and write the ARM program counter.

Erratum 550: The LFSR-based pseudo-random number generator for the GSM A5/1 encryption core was attached as an ARM coprocessor.

Erratum 551: A collection of subsystem runtime performance monitors was inserted into the test article.

Erratum 552: An extra I/O pin was added to the test article to force the VGA subsystem into high-resolution mode.

Erratum 553: An extra I/O pin was added to the test article to change the order of the SVD subsystem results.

Erratum 554: An extra I/O pin was added to force the Memory Controller subsystem into passthrough mode.

Erratum 555: One extra instruction and two extra coprocessor registers were added to the ARM subsystem in the test article.

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 1.2 I/O Description

Errata 552, 553, and 554: The following I/O pins were added to the test article:

Table 1.1: Chip I/O Signals (Added)

Signal	In/Out	Width	Description
Memory Controller			
MEM_PASS_MODE	In	1	Force Memory subsystem into passthrough mode
VGA			
HIREZ_MODE	In	1	Force VGA subsystem into high-resolution mode
Other			
SVDDL_MODE	In	1	Force SVD subsystem to change order of results

Erratum 762: The following I/O signal width was corrected:

Table 1.2: Chip I/O Signals (Corrected)

Signal	In/Out	Width	Description
Memory Controller			
MEM_ADDR	Out	28	Memory address

## 2 | ARM Subsystem Errata

### 2.1 Errata

Erratum 549: The JTAG interface can be used to read and write the ARM program counter.

Erratum 550: A GSM A5/1 cypher core was attached to the ARM coprocessor.

Erratum 551: A collection of subsystem run-time performance monitors was inserted into the test article.

Erratum 555: One extra instruction and two extra coprocessor registers were added to the ARM subsystem in the test article.

### 2.2 I/O Description

Errata 551: The following I/O pins were added to the ARM Subsystem.

Table 2.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	Out	1	ARM processor stall signal

### 2.3 Technical Details

#### 2.3.1 ARM Core

Erratum 551: The ARM processor's fetch stall signal is connected from the ARM subsystem to the SVD subsystem. The processor stalls when it performs I/O transactions to memory. More details regarding the monitoring of the ARM processor's cpuwait signal can be found in Chapter 3.

Erratum 555: A new bounded multiply operation MULB has been added to the ARM instruction set. The regular MUL instruction treats the <Rd> opcode bits [15:12] as reserved, and requires that they be set to zero. When <Rd> is non-zero, the processor instead executes the MULB instruction, and uses Rd as a bound on the result. If the product exceeds the bound, the bound is returned instead of the product. In all other respects the MUL and MULB instructions are identical, and MULB reduces to MUL when <Rd> is zero.

**MULBcdS** regD, RegA, RegB, RegC

Multiply RegA and RegB, bounded by RegC, and place into RegD. If RegC is r0, no bound is used, and the operation is MULcdS.

$$\text{RegD} = (\text{RegA} \times \text{RegB}) > \text{RegC} ? \text{RegC} : (\text{RegA} \times \text{RegB})$$

Execute only if cd is true.

Set flags if S is specified.

#### 2.3.2 ARM Control Registers

The ARM VL86C020 and derivative processors include control registers used for cache control and device identification. These control registers are accessible as built-in Coprocessor 15.

Erratum 555: Coprocessor 15 Control Register 0 (Identity Register) can be written through the JTAG register DATA\_OUT, and read by the ARM. This allows the user to override the standard ARM v2 processor identification code.

Erratum 555: Coprocessor 15 Control Register 1 (Cache Flush) is now an actual 32-bit register that can be written by the ARM, and read through the JTAG register DATA\_IN. This provides a debug mechanism, allowing the user to share data on the JTAG port. Writing to Coprocessor 15 Control Register 1 still forces a cache flush as expected.

### 2.3.3 Wishbone Debug Interface

Erratum 549: The JTAG interface was extended to permit reading and writing the ARM program counter. The new JTAG instructions are shown in Table 2.2.

Table 2.2: AVR JTAG Instruction Register (Added)

Address	Name	Data Width	Description
0x10	BSCANO	32	Write ARM program counter
0x11	BSCANI	32	Read ARM program counter

### 2.3.4 GSM A5/1 Stream Cypher

Erratum 550: This entire subsection has been added as an erratum.

A GSM A5/1 stream cypher core is attached to the ARM core through Coprocessor 15. This core is used to create a keystream that can be used to encrypt plain text. The cypher core implements GSM A5/1 to produce a running keystream by XORing the most significant bits of 3 Linear Feedback Shift Registers (LFSRs). The core can reset its contents and then accept a 64-bit externally supplied secret session key and a 22-bit frame number to prepare for keystream generation. During the preparation process, the least significant bit of each LFSR is XORed with a corresponding bit from the secret session key, and after that with a corresponding bit from the frame number. During this preparation phase, all LFSRs operate continuously with regular clocking. The eight possible modes of the 3-bit address port can be used for the purpose of loading the secret session key and frame number.

Once the secret session key and frame number have been loaded into the LFSRs, the address lines can be used to place the core in keystream generation mode to produce a pair of 114-bit keystreams. These keystreams are grouped into 32-bit words, and accessed by the ARM core through the Coprocessor 15 interface.

During the A5/1 keystream generation phase the core uses a combination of the three LFSRs operated in an irregular clocking scheme to iteratively generate 3 separate sequences of bits, which are then XORed to generate a bit of keystream per clock cycle. The A5/1 LFSR parameters are shown in Table 2.3. LFSRs whose clocking bit equals the majority value of all clocking bits will shift their contents. If any of the LFSRs does not match the majority value, it is stalled until its clock bit equals the majority value.

Table 2.3: GSM A5/1 Parameters

LFSR	Length	Feedback Polynomial	Clocking Bit
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8
2	22	$x^{22} + x^{21} + 1$	10
3	23	$x^{23} + x^{22} + x^{21} + x^8 + 1$	10

The A5/1 algorithm requires three LFSRs of bit lengths 19, 22, and 23, but the design implements them using three 32 bit registers, with the lengths of the LFSRs being initialized prior to keystream generation. Consequently, each bit holding and bit manipulating function associated with each bit position in the LFSRs is designed as a generic unit-block circuit. Through the use of several control signals, a unit-block can operate in regular or



irregular clocking modes and can appropriately XOR its contents with a value received from polynomial evaluation performed on more significant bits. This means that the core can also be used as a pseudo-random number generator, by initializing the LFSR lengths, polynomials, and clocking bits.

The core is connected to the ARM core via a 32-bit coprocessor interface. It is the responsibility of the software on the ARM core to appropriately load and use the two 114-bit keystream pairs. In addition, the module has a 3-bit address port and a read/write strobe signal interface with the coprocessor. Once a keystream has been generated, the plaintext encryption can be done outside the core.

The A5/1 core is initialized by writing to Coprocessor 15 register CR6. Keystream data is obtained from the core by reading from Coprocessor 15 register CR8. These registers use self-incrementing counters, so data must always be written to or read from them in groups of eight words. The initialization data sequence is presented in Table 2.4, and the keystream data sequence is presented in Table 2.5.

Table 2.4: Initialization Sequence: Coprocessor 15 Register CR6

Index	Bits	Description
0	[7:0]	LFSR 0 length
0	[15:8]	LFSR 1 length
0	[23:16]	LFSR 2 length
0	[31:24]	Reserved
1	[31:0]	LFSR 0 polynomial
2	[31:0]	LFSR 1 polynomial
3	[31:0]	LFSR 2 polynomial
4	[3:0]	LFSR 0 clocking bit
4	[7:4]	LFSR 1 clocking bit
4	[11:8]	LFSR 2 clocking bit
4	[31:12]	Reserved
5	[31:0]	LFSR 0 session key
6	[31:0]	LFSR 1 session key
7	[21:0]	LFSR 2 session key

Table 2.5: Keystream Sequence: Coprocessor 15 Register CR8

Index	Description
0	Keystream 0 bits [31:0]
1	Keystream 0 bits [63:32]
2	Keystream 0 bits [95:64]
3	Keystream 0 bits [127:96]
4	Keystream 1 bits [31:0]
5	Keystream 1 bits [63:32]
6	Keystream 1 bits [95:64]
7	Keystream 1 bits [127:96]

## 3 | SVD Subsystem Errata

### 3.1 Errata

Erratum 551: A collection of subsystem run-time performance monitors was inserted into the test article.

Erratum 553: An extra I/O pin was added to the test article to change the order of the SVD subsystem results.

### 3.2 Block Diagram

Erratum 551: The following block diagram reflects the modifications made to the SVD subsystem.

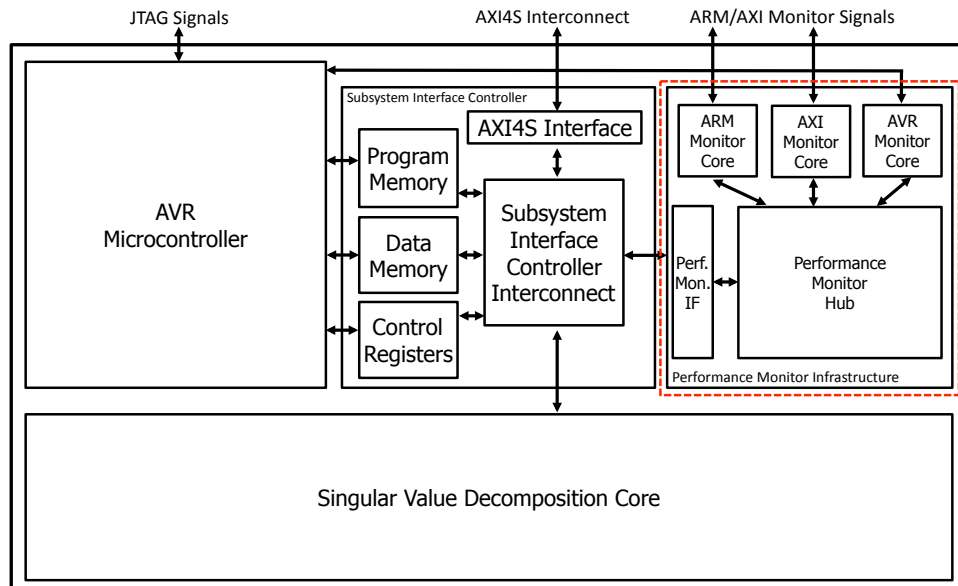


Figure 3.1: SVD Subsystem Block Diagram

### 3.3 I/O Description

Errata 551 and 553: The following I/O pins were added to the SVD subsystem.

Table 3.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	In	1	ARM processor stall signal
axi_ports_empty	In	8	AXI Interconnect Input FIFO empty signal
axi_ports_full	In	8	AXI Interconnect Input FIFO full signal
axi_ports_valid	In	8	AXI Interconnect Input FIFO valid signal
axi_ports_ready	In	8	AXI Interconnect Input FIFO ready signal
SVDDL_MODE	In	1	Force SVD subsystem to change order of results

## 3.4 Technical Details

### 3.4.1 SVD Reordering

Erratum 553: If the device I/O SVDDL\_MODE pin is driven high, the selection of  $\Sigma$  and U vectors is swapped when read back from the core. Details are shown in Table 3.2.

Table 3.2: SVD Addressing and Control

Address Bits [32:10]	[9:8]	[7:6]	[5:1]	[0]	Description
SVDDL_MODE = 0					
0010 0100 0000 0000 00	00	01	[row]	0	Read $\Sigma$ Vector S[31:0]
0010 0100 0000 0000 00	00	01	[row]	1	Read $\Sigma$ Vector S[63:32]
0010 0100 0000 0000 00	00	10	[row]	0	Read Left Singular Vector U[31:0]
0010 0100 0000 0000 00	00	10	[row]	1	Read Left Singular Vector U[63:32]
SVDDL_MODE = 1					
0010 0100 0000 0000 00	00	01	[row]	0	Read Left Singular Vector U[31:0]
0010 0100 0000 0000 00	00	01	[row]	1	Read Left Singular Vector U[63:32]
0010 0100 0000 0000 00	00	10	[row]	0	Read $\Sigma$ Vector S[31:0]
0010 0100 0000 0000 00	00	10	[row]	1	Read $\Sigma$ Vector S[63:32]

### 3.4.2 Performance Monitors Infrastructure

Erratum 551: This entire subsection has been added as an erratum.

The performance monitor infrastructure provides run-time system information. The information can be collected and used by a designer to better understand the system performance under various loads and conditions. The system uses individual cores to monitor the ARM processor, AVR processor, and AXI4S interconnect. A designer can enable or disable monitoring and capture or reset each monitor core's data. The monitoring infrastructure is composed of the following blocks:

- Performance Monitor Interface
- Performance Monitor Hub
- ARM Performance Monitor Core
- AVR Performance Monitor Core
- AXI4S Interconnect Monitor Core

The ARM subsystem and the AXI4S interconnect are separate from the SVD subsystem, but their monitoring cores reside within the SVD subsystem. Figure 3.1 shows the performance monitor infrastructure integrated into the SVD subsystem, including the subsystem I/O ports added for external monitoring of the ARM subsystem and AXI4S interconnect.

#### 3.4.2.1 Performance Monitor Interface

The system interacts with the Performance Monitor through the Performance Monitor Interface. An additional port was added to Subsystem Interface Controller (SIC) interconnect. This port connects to the Performance Monitor Interface at address 0x25000000. The interface also adds separate 16-element deep FIFOs on the transmit and receive ports to buffer commands and data going to and from the system.

#### 3.4.2.2 Performance Monitor Hub

The Performance Monitor Hub aggregates commands from the system and passes them on to the specified performance monitor core. Table 3.3 defines the supported commands.

Table 3.3: Performance Monitor Commands

Command	Description
0x0	Retrieve all data from all performance monitors
0x1	Retrieve all data from a specific performance monitor
0x2	Retrieve a specific data word from all performance monitors
0x3	Retrieve a specific data word from one performance monitor
0x4	Reset data for all performance monitors
0x5	Reset data for a specific performance monitor
0x6	Enable data collection for all performance monitors
0x7	Enable data collection for a specific performance monitor
0x8	Disable data collection for all performance monitors
0x9	Disable data collection for a specific performance monitor

Table 3.4 enumerates the performance monitor cores. These numbers can be combined with commands to designate a specific performance monitor.

Table 3.4: Performance Monitor Cores Numeric Representation

Number	Core Name
0	AVR Processor Performance Monitor
1	ARM Processor Performance Monitor
2	AXI Interconnect Performance Monitor

Table 3.5 describes the Performance Monitor Hub Command Register at address 0x25000000.

Table 3.5: Performance Monitor Hub Command Register

Bit number	Access	Description
[31:12]	—	Reserved
[11:8]	w	Monitor number (Table 3.4)
[7:4]	—	Reserved
[3:0]	w	Command (Table 3.3)

After a command is issued, the resulting data can be read from address 0x25000000. The data returned depends on the command that was issued. The first word of data indicates how many monitors are included in the results. Then for each monitor, the number of data words, followed by the actual data words are returned. A simple C program with a double-nested loop can be used to iterate over each monitor and then over each datum.

### 3.4.2.3 ARM Performance Monitor Core

The ARM performance monitor core receives input signal `arm_cpawait`. When the `arm_cpawait` signal is high, the ARM processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `arm_cpawait` signal is active. Combined with the total run-time of the ARM subsystem, a user can quickly understand the utilization of the processor core. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ARM monitor includes two 64-bit timers. Timer 0 measures the idle time when `arm_cpawait` is asserted. Timer 1 measures the active run time, when `arm_cpawait` is not asserted. The monitor data is described in Table 3.6.

Table 3.6: ARM Performance Monitor Core's Data Order

Word	Description
0	Timer 0: ARM processor idle timer [31:0] data
1	Timer 0: ARM processor idle timer [63:32] data
2	Timer 1: ARM processor run timer [31:0] data
3	Timer 1: ARM processor run timer [63:32] data

#### 3.4.2.4 AXI4S Interconnect Performance Monitor Core

The AXI4S interconnect performance monitor core receives inputs `axi_port_empty`, `axi_port_full`, `axi_port_valid`, and `axi_port_ready`. These signals reflect the AXI4S interconnect input FIFO status. The ports in the TA1B interconnect each have FIFOs to buffer incoming data. The FIFO status is useful for understanding the utilization of the interconnect and the load distribution of an application on the system. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AXI4S monitor data includes one 32-bit word indicating the status of the input FIFOs. The AXI4S interconnect has 8 ports. The status information is divided into four groups as shown in Table 3.7. Within each group the bit position corresponds to the port number.

Table 3.7: AXI4S Interconnect Status Register

Bit number	Access	Description
[31:24]	r	AXI4S input FIFO ready signals
[23:16]	r	AXI4S input FIFO valid signals
[15:8]	r	AXI4S input FIFO full signals
[7:0]	r	AXI4S input FIFO empty signals

#### 3.4.2.5 AVR Performance Monitor Core

The AVR performance monitor core receives inputs `avr_cpuwait`, `avr_pc`, and `avr_inst`. When the `avr_cpuwait` signal is high, the AVR processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `avr_cpuwait` signal is active. Combined with the total run-time of the AVR subsystem, a user can quickly understand the utilization of the processor core. The monitor can also capture the current value of the program counter and the current instruction that is being executed. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AVR monitor includes two 64-bit timers and two 32-bit words for the program counter and current instruction. Timer 0 measures the idle time when the `avr_cpuwait` signal is asserted. Timer 1 measures the active run time, when the `avr_cpuwait` signal is not asserted. The monitor data is described in Table 3.8.

Table 3.8: AVR Performance Monitor Core's Data Order

Word	Description
0	Timer 0: AVR processor idle timer [31:0] data
1	Timer 0: AVR processor idle timer [63:32] data
2	Timer 1: AVR processor run timer [31:0] data
3	Timer 1: AVR processor run timer [63:32] data
4	AVR processor program counter
5	AVR processor instruction register

# 4 | Memory Controller Subsystem Errata

## 4.1 Errata

Erratum 554: An extra I/O pin was added to force the Memory Controller subsystem into passthrough mode.

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 4.2 I/O Description

Errata 554: The following I/O pins was added to the test article:

Table 4.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
MEM_PASS_MODE	In	1	Force Memory subsystem into passthrough mode

Erratum 762: The following I/O signal width was corrected:

Table 4.2: Subsystem I/O Signals (Changed)

Signal	In/Out	Width	Description
MEM_ADDR	Out	28	Off-chip memory address. Provides the base address (or the start address in case of a burst) of the data to be accessed.

## 4.3 Technical Details

### 4.3.1 Passthrough Mode

Erratum 554: The Memory Controller subsystem can be forced into Passthrough mode by driving the device I/O MEM\_PASS\_MODE pin high. The documented method of entering Passthrough mode by asserting the ACK signal and holding the two MSBs of MEM\_DATA\_IN high also remains valid.

## **5 | SPI Subsystem Errata**

### **5.1 Errata**

No errata exist for this subsystem.

## 6 | I2C Subsystem Errata

### 6.1 Errata

No errata exist for this subsystem.



## 7 | UART Subsystem Errata

### 7.1 Errata

No errata exist for this subsystem.

## 8 | VGA Subsystem Errata

### 8.1 Errata

Erratum 552: An extra I/O pin was added to the test article to force the VGA subsystem into high-resolution mode.

### 8.2 I/O Description

Erratum 552: The following I/O pin was added to the SVD Subsystem.

Table 8.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
HIREZ_MODE	In	1	Force VGA subsystem into high-resolution mode

### 8.3 Technical Details

Erratum 552: If the device I/O HIREZ\_MODE pin is driven high, the VGA subsystem is forced into high-resolution 800 × 600 mode, regardless of register settings.

# 9 | AXI4 Interconnect Errata

## 9.1 Errata

Erratum 548: The AXI4S interconnect uses Round Robin arbitration with priority given to higher port numbers.

Erratum 551: A collection of subsystem run-time performance monitors was inserted into the test article.

## 9.2 I/O Description

Errata 551: The following I/O pins were added to the AXI4 Interconnect.

Table 9.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
axi_ports_empty	Out	8	AXI Interconnect Input FIFO empty signal
axi_ports_full	Out	8	AXI Interconnect Input FIFO full signal
axi_ports_valid	Out	8	AXI Interconnect Input FIFO valid signal
axi_ports_ready	Out	8	AXI Interconnect Input FIFO ready signal

## 9.3 Technical Details

### 9.3.1 Crossbar Switch

Erratum 551: Each input port's FIFO status signals in the AXI interconnect are routed to the SVD subsystem. More details regarding the monitor of the FIFO signals can be found in Chapter 3.

### 9.3.2 Arbitration

Erratum 548: If multiple requests for the same output port reach the arbiter during the same clock cycle, priority is given to the request with the highest port number. All other requests will be enqueued and prioritized from highest to lowest port number.

---

# **ITAG PHASE 1 THRUST 3A TEST ARTICLE ANSWER KEY**

Information Sciences Institute  
University of Southern California

**FOR IRIS INTERNAL USE**

Modified: November 7, 2012

<b>0</b>	<b>Preface</b>	<b>3</b>
0.1	Overview . . . . .	3
0.2	Errata List . . . . .	3
<b>1</b>	<b>System Errata</b>	<b>4</b>
1.1	Errata . . . . .	4
1.2	Features . . . . .	4
1.3	Block Diagram . . . . .	4
1.4	I/O Description . . . . .	4
<b>2</b>	<b>ARM Subsystem Errata</b>	<b>5</b>
2.1	Errata . . . . .	5
2.2	I/O Description . . . . .	5
2.3	Technical Details . . . . .	5
2.3.1	ARM Core . . . . .	5
<b>3</b>	<b>Memory Controller Subsystem Errata</b>	<b>6</b>
3.1	Errata . . . . .	6
3.2	I/O Description . . . . .	6
<b>4</b>	<b>Peripheral Subsystem Errata</b>	<b>7</b>
4.1	Errata . . . . .	7
4.2	Technical Details . . . . .	7
<b>5</b>	<b>Cryptographic Subsystem Errata</b>	<b>9</b>
5.1	Errata . . . . .	9
5.2	Block Diagram . . . . .	9
5.3	I/O Description . . . . .	9
5.4	Technical Details . . . . .	9
5.4.1	Subsystem Interface Controller . . . . .	9
5.4.2	Crypto Core . . . . .	10
5.4.3	Performance Monitors Infrastructure . . . . .	10
<b>6</b>	<b>AXI4 Interconnect Errata</b>	<b>13</b>
6.1	Errata . . . . .	13
6.2	Block Diagram . . . . .	13
6.3	Technical Details . . . . .	13
6.3.1	Crossbar Switch . . . . .	13
6.3.2	Arbitration . . . . .	13

# 0 | Preface

## 0.1 Overview

The ITAG Phase 1 Thrust 3A Test Article (TA3A) is a soft IP System-on-Chip (SoC) developed by USC Information Sciences Institute in support of the DARPA Integrity and Reliability of Integrated Circuits (IRIS) Thrust 3A. This soft IP is intended for implementation in an ASIC.

This document describes differences between the delivered TA3A test article and the corresponding datasheet released to IRIS performers.

## 0.2 Errata List

Each difference between the TA3A test article and datasheet is listed below and numbered according to the ITAG internal tracking number.

- 591:** Extra AXI4S Interconnect Port. An extra port was added to the AXI4S interconnect for system expansion. This erratum is described in Section 1.3, 6.2, and 6.3.1.
- 592:** Cryptographic Subsystem Bypass Mode. The encryption feature can be disabled. This erratum is described in Section 5.4.1.1 and 5.4.2.
- 593:** Network Routing Arbitration. The AXI4S interconnect uses Round Robin arbitration with priority given to higher port numbers. This erratum is described in Section 6.3.2.
- 595:** Performance Monitors. A collection of subsystem runtime performance monitors was inserted into the test article. This erratum is described in Chapter 5 (sections 5.2, 5.3, and 5.4.3) and sections 2.2 and 2.3.1.
- 601:** Writable UART Counters. Writable UART counters were inserted into the test article to allow runtime baud rate adjustments. This erratum is described in Section 1.2 and 4.2.
- 762:** Address Pin Count Mismatch. The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24. This erratum is described in sections 1.4 and 3.2.

# 1 | System Errata

## 1.1 Errata

Erratum 591: An extra port was added to the AXI4S interconnect for system expansion.

Erratum 601: Writable UART counters were inserted into the test article to allow runtime baud rate adjustments.

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 1.2 Features

- Erratum 601: UART baud rates from 300 to 4,608,000

## 1.3 Block Diagram

Erratum 591: An extra port was added to the AXI4S interconnect for system expansion.

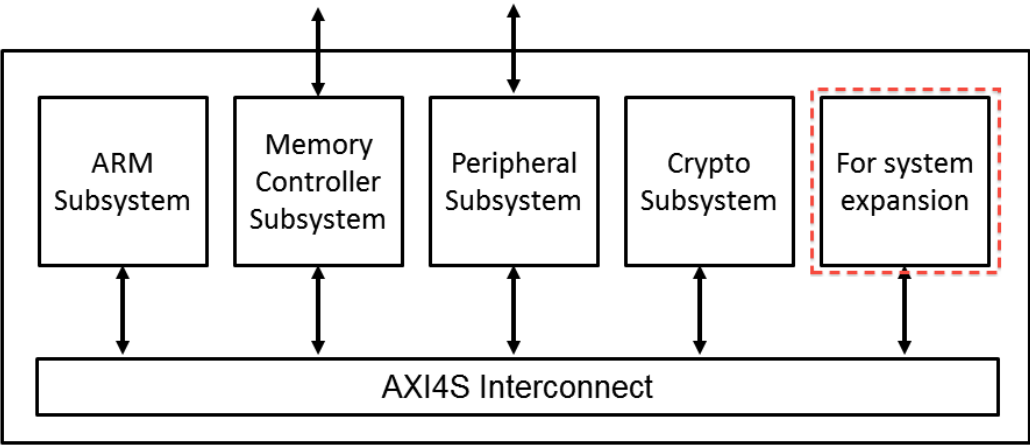


Figure 1.1: High-level block diagram of the TA3A System-on-Chip

## 1.4 I/O Description

Erratum 762: The following I/O signal width was corrected:

Table 1.1: Chip I/O Signals (Corrected)

Signal	In/Out	Width	Description
Memory Controller			
MEM_ADDR	Out	28	Memory address

## 2 | ARM Subsystem Errata

### 2.1 Errata

Erratum 595: A collection of subsystem run-time performance monitors was inserted into the test article.

### 2.2 I/O Description

Erratum 595: The following I/O pins were added to the ARM Subsystem.

Table 2.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	Out	1	ARM processor stall signal

### 2.3 Technical Details

#### 2.3.1 ARM Core

Erratum 595: The ARM processor's fetch stall signal is connected from the ARM subsystem to the Cryptographic subsystem. The processor stalls when it performs I/O transactions to memory. More details regarding the monitoring of the ARM processor's cpuwait signal can be found in Chapter 5.



# 3 | Memory Controller Subsystem Errata

## 3.1 Errata

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 3.2 I/O Description

Erratum 762: The following I/O signal width was corrected:

Table 3.1: Subsystem I/O Signals (Corrected)

Signal	In/Out	Width	Description
MEM_ADDR	Out	28	Off-chip memory address. Provides the base address (or the start address in case of a burst) of the data to be accessed.

# 4 | Peripheral Subsystem Errata

## 4.1 Errata

Erratum 601: Writable UART counters were inserted into the test article to allow runtime baud rate adjustments.

## 4.2 Technical Details

Erratum 601: The UART supports operations to receive and transmit data, to get and set the baud rate, to get the FIFO status, and to acquire, check, or release a mutex. The operation requested is determined by the read or write address from Table 4.1.

Table 4.1: UART Address Summary

Address	Description
0x22000000	Normal Operation
0x22000004	Get/Set Baud Low
0x22000008	Get/Set Baud High
0x2200000C	Get FIFO Status
0x22000010	Check Mutex
0x22000010	Acquire Mutex
0x22000020	Release Mutex

Erratum 601: The UART baud rate is controlled by two 32-bit registers. The low 12 bits at address 0x22000004 set the baud frequency and the low 16 bits at address 0x22000008 set the baud limit. These registers together set two internal counters that configure the baud clock.

Erratum 601: The UART default baud rate is 57,600 bps. Table 4.2 shows the baud rate settings to use if the system clock frequency is 50 MHz.

Table 4.2: UART Settings

Baud Rate	baud_freq	baud_limit
300	0x0003	0xF421
600	0x0003	0x7A0F
1,200	0x0003	0x3D06
2,400	0x0006	0x3D03
4,800	0x000C	0x3CFD
9,600	0x0018	0x3CF1
14,400	0x0024	0x3CE5
19,200	0x0030	0x3CD9
28,800	0x0048	0x3CC1
38,400	0x0060	0x3CA9
56,000	0x001C	0x0C19
<b>57,600<sup>†</sup></b>	<b>0x0090</b>	<b>0x3C79</b>
115,200	0x0120	0x3BE9
128,000	0x0040	0x0BF5
153,600	0x0180	0x3B89
230,400	0x0240	0x3AC9
256,000	0x0080	0x0BB5
460,800	0x0480	0x3889
921,600	0x0900	0x3409
1,382,400	0x0D80	0x2F89
2,304,000	0x0480	0x07B5
4,608,000	0x0900	0x0335

<sup>†</sup> Default baud rate

Erratum 601: The baud settings in Table 4.2 can be calculated from the desired baud rate as follows:

$$Baud\_freq = \frac{16 \times baud\_rate}{gcd(system\_clock\_freq, 16 \times baud\_rate)}$$

$$Baud\_limit = \frac{system\_clock\_freq}{gcd(system\_clock\_freq, 16 \times baud\_rate)} - baud\_freq$$

## 5 | Cryptographic Subsystem Errata

### 5.1 Errata

Erratum 592: The encryption feature can be disabled.

Erratum 595: A collection of subsystem run-time performance monitors was inserted into the test article.

### 5.2 Block Diagram

Errata 595: The following block diagram reflects the modifications made to the Cryptographic subsystem.

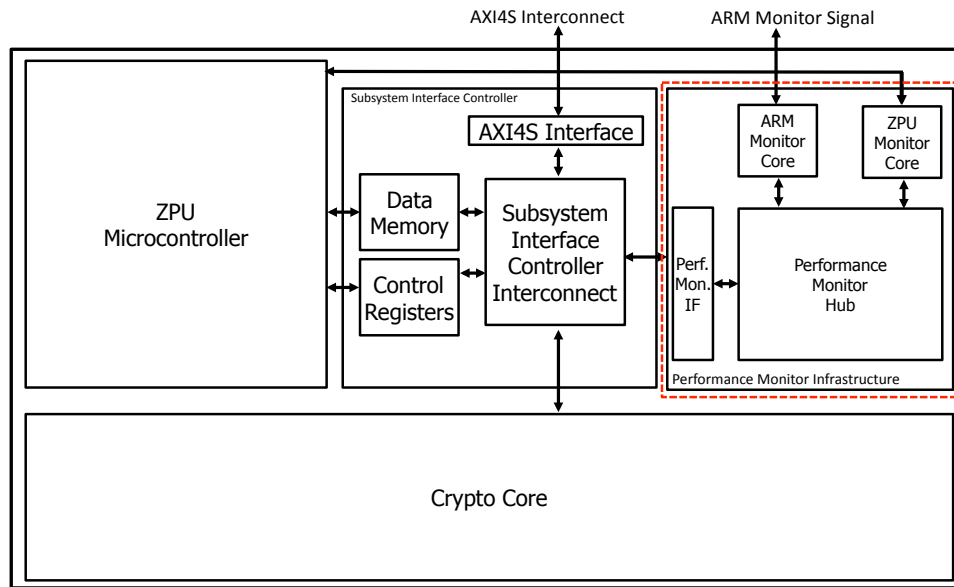


Figure 5.1: Cryptographic Subsystem Block Diagram

### 5.3 I/O Description

Errata 595: The following I/O pins were added to the Cryptographic Subsystem.

Table 5.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	In	1	ARM processor stall signal

### 5.4 Technical Details

#### 5.4.1 Subsystem Interface Controller

##### 5.4.1.1 SIC Control Registers

Erratum 592: The following select bits were corrected. When the select bits are set to either 10 or 11 for bypass mode, data encryption is disabled.

Table 5.2: Remote Request Register (Corrected)

Bit number	Access	Description
[31:18]	—	Reserved
[17:16]	r/w	Crypto Core algorithm select (00 = JH, 01 = Blake, 10/11 = Bypass) [default:00]

### 5.4.2 Crypto Core

Erratum 592: Before using the Crypto core to generate a hash for a message, the user must select the proper algorithm for data encryption. Write 0x00010000 for Blake, 0x00000000 for JH, or 0x00020000 or 0x00030000 for bypass to the SIC Remote Request Register at address 0x32000010. JH is selected by default.

### 5.4.3 Performance Monitors Infrastructure

Erratum 595: This entire subsection has been added as an erratum.

The performance monitor infrastructure provides run-time system information. The information can be collected and used by a designer to better understand the system performance under various loads and conditions. The system uses individual cores to monitor the ARM processor and ZPU processor. A designer can enable or disable monitoring and capture or reset each monitor core's data. The monitoring infrastructure is composed of the following blocks:

- Performance Monitor Interface
- ARM Performance Monitor Core
- Performance Monitor Hub
- ZPU Performance Monitor Core

The ARM subsystem is separate from the Cryptographic subsystem, but its monitoring core resides within the Cryptographic subsystem. Figure 5.1 shows the performance monitor infrastructure integrated into the Cryptographic subsystem, including the subsystem I/O port added for external monitoring of the ARM subsystem.

#### 5.4.3.1 Performance Monitor Interface

The system interacts with the Performance Monitor through the Performance Monitor Interface. An additional port was added to Subsystem Interface Controller (SIC) interconnect. This port connects to the Performance Monitor Interface at address 0x34000000. The interface also adds separate 16-element deep FIFOs on the transmit and receive ports to buffer commands and data going to and from the system.

#### 5.4.3.2 Performance Monitor Hub

The Performance Monitor Hub aggregates commands from the system and passes them on to the specified performance monitor core. Table 5.3 defines the supported commands.

Table 5.3: Performance Monitor Commands

Command	Description
0x0	Retrieve all data from all performance monitors
0x1	Retrieve all data from a specific performance monitor
0x2	Retrieve a specific data word from all performance monitors
0x3	Retrieve a specific data word from one performance monitor
0x4	Reset data for all performance monitors
0x5	Reset data for a specific performance monitor
0x6	Enable data collection for all performance monitors
0x7	Enable data collection for a specific performance monitor
0x8	Disable data collection for all performance monitors
0x9	Disable data collection for a specific performance monitor

Table 5.4 enumerates the performance monitor cores. These numbers can be combined with commands to designate a specific performance monitor.

Table 5.4: Performance Monitor Cores Numeric Representation

Number	Core Name
0	ZPU Processor Performance Monitor
1	ARM Processor Performance Monitor

Table 5.5 describes the Performance Monitor Hub Command Register at address 0x34000000.

Table 5.5: Performance Monitor Hub Command Register

Bit number	Access	Description
[31:12]	—	Reserved
[11:8]	w	Monitor number (Table 5.4)
[7:4]	—	Reserved
[3:0]	w	Command (Table 5.3)

After a command is issued, the resulting data can be read from address 0x34000000. The data returned depends on the command that was issued. The first word of data indicates how many monitors are included in the results. Then for each monitor, the number of data words, followed by the actual data words are returned. A simple C program with a double-nested loop can be used to iterate over each monitor and then over each datum.

#### 5.4.3.3 ARM Performance Monitor Core

The ARM performance monitor core receives input signal `arm_cpawait`. When the `arm_cpawait` signal is high, the ARM processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `arm_cpawait` signal is active. Combined with the total run-time of the ARM subsystem, a user can quickly understand the utilization of the processor core. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ARM monitor includes two 64-bit timers. Timer 0 measures the idle time when `arm_cpawait` is asserted. Timer 1 measures the active run time, when `arm_cpawait` is not asserted. The monitor data is described in Table 5.6.

Table 5.6: ARM Performance Monitor Core's Data Order

Word	Description
0	Timer 0: ARM processor idle timer [31:0] data
1	Timer 0: ARM processor idle timer [63:32] data
2	Timer 1: ARM processor run timer [31:0] data
3	Timer 1: ARM processor run timer [63:32] data

#### 5.4.3.4 ZPU Performance Monitor Core

The ZPU performance monitor core receives inputs `zpu_cpawait`, `zpu_pc`, and `zpu_inst`. When the `zpu_cpawait` signal is high, the ZPU processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `zpu_cpawait` signal is active. Combined with the total run-time of the ZPU subsystem, a user can quickly understand the utilization of the processor core. The monitor can also capture the current value of the program counter and the current instruction that is being executed. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ZPU monitor includes two 64-bit timers and two 32-bit words for the program counter and current instruction. Timer 0 measures the idle time when the `zpu_cpuwait` signal is asserted. Timer 1 measures the active run time, when the `zpu_cpuwait` signal is not asserted. The monitor data is described in Table 5.7.

Table 5.7: ZPU Performance Monitor Core's Data Order

Word	Description
0	Timer 0: ZPU processor idle timer [31:0] data
1	Timer 0: ZPU processor idle timer [63:32] data
2	Timer 1: ZPU processor run timer [31:0] data
3	Timer 1: ZPU processor run timer [63:32] data
4	ZPU processor program counter
5	ZPU processor instruction register

## 6 | AXI4 Interconnect Errata

### 6.1 Errata

Erratum 591: An extra port was added to the AXI4S interconnect for system expansion.

Erratum 593: The AXI4S interconnect uses Round Robin arbitration with priority given to higher port numbers.

### 6.2 Block Diagram

Erratum 591: Extra port number 4 was added to the TA3A AXI4S interconnect as shown in Figure 6.1.

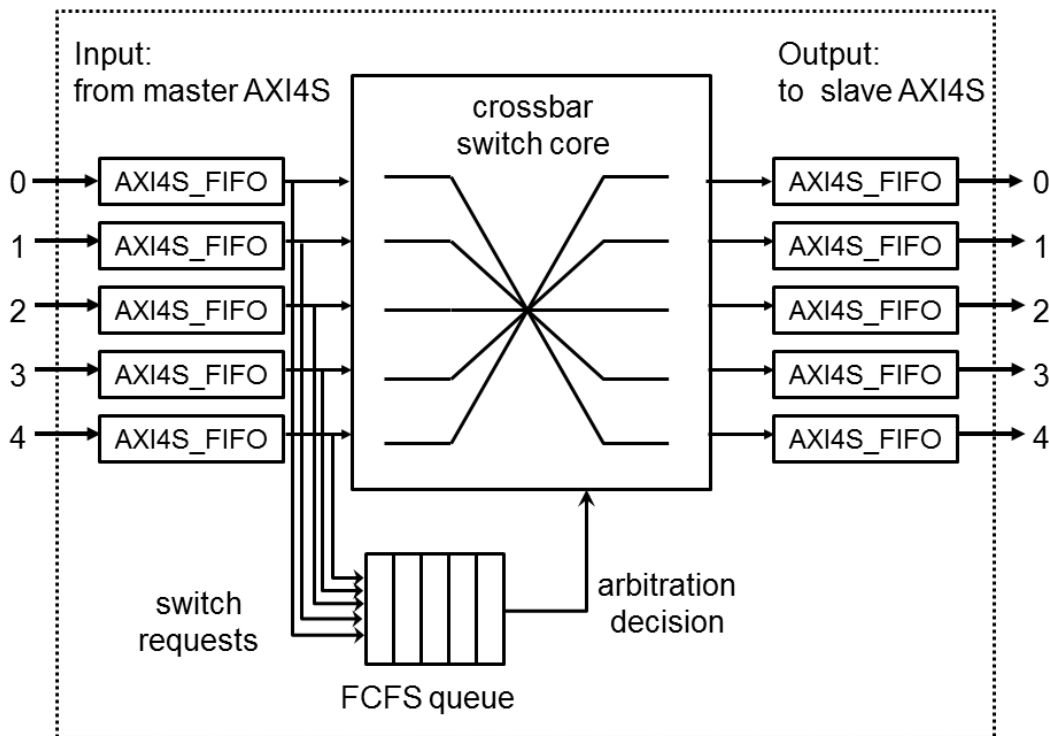


Figure 6.1: Block diagram of AXI4S switch

### 6.3 Technical Details

#### 6.3.1 Crossbar Switch

Erratum 591: Extra port number 4 was added to the AXI4S Interconnect, so the crossbar is now a five port switch.

#### 6.3.2 Arbitration

Erratum 593: If multiple requests for the same output port reach the arbiter during the same clock cycle, priority is given to the request with the highest port number. All other requests will be enqueued and prioritized from highest to lowest port number.



---

# **ITAG PHASE 1 THRUST 3B TEST ARTICLE ANSWER KEY**

Information Sciences Institute  
University of Southern California

**FOR IRIS INTERNAL USE**

Modified: November 7, 2012

<b>0 Preface</b>	<b>4</b>
0.1 Overview	4
0.2 Errata List	4
<b>1 System Errata</b>	<b>5</b>
1.1 Errata	5
1.2 Features	5
1.3 Block Diagram	5
1.4 I/O Description	5
<b>2 ARM Subsystem Errata</b>	<b>6</b>
2.1 Errata	6
2.2 I/O Description	6
2.3 Technical Details	6
2.3.1 ARM Core	6
2.3.2 GSM A5/1 Stream Cypher	6
<b>3 Smart Memory Subsystem Errata</b>	<b>9</b>
3.1 Errata	9
3.2 I/O Description	9
<b>4 Maintenance Subsystem Errata</b>	<b>10</b>
4.1 Errata	10
4.2 Features	10
4.3 Block Diagram	10
4.4 I/O Description	10
4.5 Technical Details	11
4.5.1 Performance Monitors Infrastructure	11
<b>5 Peripheral Subsystem Errata</b>	<b>16</b>
5.1 Errata	16
5.2 Technical Details	16
<b>6 Cryptographic Subsystem Errata</b>	<b>18</b>
6.1 Errata	18
6.2 Features	18
6.3 I/O Description	18
6.4 Technical Details	18
6.4.1 ZPU Core	18
6.4.2 Subsystem Interface Controller	18
6.4.3 Crypto Core	19
<b>7 Hardware Control Subsystem Errata</b>	<b>20</b>
7.1 Errata	20
7.2 Technical Details	20
<b>8 AVR Subsystem Errata</b>	<b>21</b>
8.1 Errata	21
<b>9 On-Chip Memory Subsystem Errata</b>	<b>22</b>
9.1 Errata	22
<b>10 JTAG Subsystem Errata</b>	<b>23</b>
10.1 Errata	23
10.2 Technical Details	23

**11 AXI4 Mesh Interconnect Errata** **24**

11.1 Errata . . . . . 24

11.2 I/O Description . . . . . 24

11.3 Technical Details . . . . . 24

# 0 | Preface

## 0.1 Overview

The ITAG Phase 1 Thrust 3B Test Article (TA3B) is a soft IP System-on-Chip (SoC) developed by USC Information Sciences Institute in support of the DARPA Integrity and Reliability of Integrated Circuits (IRIS) Thrust 3B. This soft IP is intended for implementation in Xilinx Virtex6 and Virtex7 FPGAs.

This document describes differences between the delivered TA3B test article and the corresponding datasheet released to IRIS performers.

## 0.2 Errata List

Each difference between the TA3B test article and datasheet is listed below and numbered according to the ITAG internal tracking number.

- 594:** GSM A5/1 Stream Cypher. A GSM A5/1 cypher core was attached to the ARM coprocessor. This erratum is described in Section 2.3.2.
- 597:** Mesh Routing Reconfiguration. Support for runtime reconfiguration of the AXI4 mesh interconnect. This erratum is described in Chapter 4 (sections 4.2, 4.3, 4.4, 4.5, and 4.5.0.1) and Section 11.3.
- 598:** Mesh Network Data Width. Changed the mesh network port width for the Hardware Control kernel to be 16 bits wide rather than 32 bits wide. This erratum is described in sections 7.2 and 11.3.
- 599:** ZPU JTAG. The ZPU processor's data memory is connected to the JTAG chain. This erratum is described in sections 6.4.1 and 10.2.
- 600:** Performance Monitors. A collection of subsystem runtime performance monitors was inserted into the test article. This erratum is described in Chapter 4 (sections 4.3, 4.4, and 4.5.1) and sections 2.2, 2.3.1, 6.3, 6.4.1, 11.2, and 11.3.
- 602:** Writable UART Counters. Writable UART counters were inserted into the test article to allow runtime baud rate adjustments. This erratum is described in sections 1.2 and 5.2.
- 686:** Cryptographic Subsystem Bypass Mode. The SHA-3 hash function can be bypassed. This erratum is described in sections 6.4.2.1 and 6.4.3.
- 753:** Skein Cryptographic Hash. The SHA-3 Skein candidate was added to the Cryptographic subsystem. This erratum is described in sections 6.2, 6.4.2.1, and 6.4.3.1.
- 762:** Address Pin Count Mismatch. The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24. This erratum is described in sections 1.4 and 3.2.

# 1 | System Errata

## 1.1 Errata

Erratum 602: Writable UART Counters. Writable UART counters were inserted into the test article to allow run-time baud rate adjustments.

Erratum 762: Address Pin Count Mismatch. The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 1.2 Features

- Erratum 602: UART baud rates from 300 to 4,608,000

## 1.3 Block Diagram

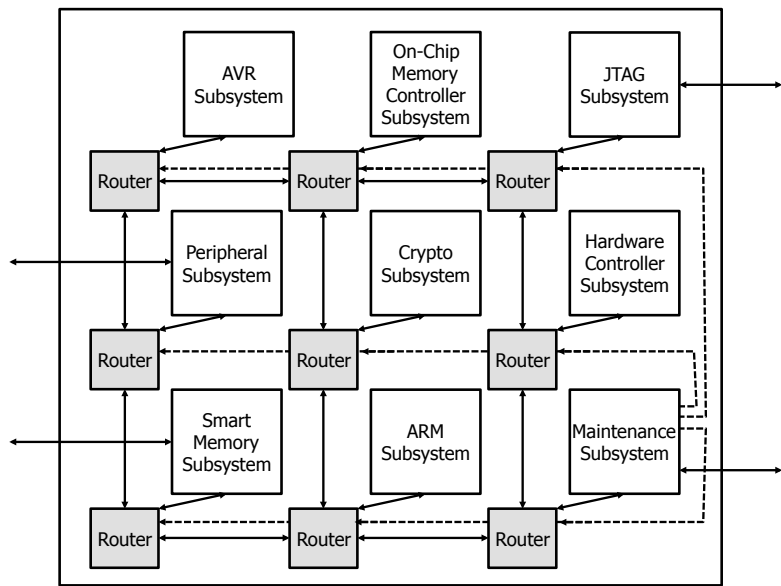


Figure 1.1: High-level block diagram of the TA3B System-on-Chip

## 1.4 I/O Description

Erratum 762: The following I/O signal width was corrected:

Table 1.1: Chip I/O Signals (Corrected)

Signal	In/Out	Width	Description
Memory Controller			
MEM_ADDR	Out	28	Memory address

## 2 | ARM Subsystem Errata

### 2.1 Errata

Erratum 594: A GSM A5/1 cypher core was attached to the ARM coprocessor.

Erratum 600: A collection of subsystem run-time performance monitors was inserted into the test article.

### 2.2 I/O Description

Erratum 600: The following I/O pins were added to the ARM Subsystem.

Table 2.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
arm_cpuwait	Out	1	ARM processor stall signal

### 2.3 Technical Details

#### 2.3.1 ARM Core

Erratum 600: The ARM processor's fetch stall signal is connected from the ARM subsystem to the SVD subsystem. The processor stalls when it performs I/O transactions to memory. More details regarding the monitoring of the ARM processor's cpuwait signal can be found in Chapter 4.

#### 2.3.2 GSM A5/1 Stream Cypher

Erratum 594: This entire subsection has been added as an erratum.

A GSM A5/1 stream cypher core is attached to the ARM core through Coprocessor 15. This core is used to create a keystream that can be used to encrypt plain text. The cypher core implements GSM A5/1 to produce a running keystream by XORing the most significant bits of 3 Linear Feedback Shift Registers (LFSRs). The core can reset its contents and then accept a 64-bit externally supplied secret session key and a 22-bit frame number to prepare for keystream generation. During the preparation process, the least significant bit of each LFSR is XORed with a corresponding bit from the secret session key, and after that with a corresponding bit from the frame number. During this preparation phase, all LFSRs operate continuously with regular clocking. The eight possible modes of the 3-bit address port can be used for the purpose of loading the secret session key and frame number.

Once the secret session key and frame number have been loaded into the LFSRs, the address lines can be used to place the core in keystream generation mode to produce a pair of 114-bit keystreams. These keystreams are grouped into 32-bit words, and accessed by the ARM core through the Coprocessor 15 interface.

During the A5/1 keystream generation phase the core uses a combination of the three LFSRs operated in an irregular clocking scheme to iteratively generate 3 separate sequences of bits, which are then XORed to generate a bit of keystream per clock cycle. The A5/1 LFSR parameters are shown in Table 2.2. LFSRs whose clocking bit equals the majority value of all clocking bits will shift their contents. If any of the LFSRs does not match the majority value, it is stalled until its clock bit equals the majority value.

Table 2.2: GSM A5/1 Parameters

LFSR	Length	Feedback Polynomial	Clocking Bit
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8
2	22	$x^{22} + x^{21} + 1$	10
3	23	$x^{23} + x^{22} + x^{21} + x^8 + 1$	10

The A5/1 algorithm requires three LFSRs of bit lengths 19, 22, and 23, but the design implements them using three 32 bit registers, with the lengths of the LFSRs being initialized prior to keystream generation. Consequently, each bit holding and bit manipulating function associated with each bit position in the LFSRs is designed as a generic unit-block circuit. Through the use of several control signals, a unit-block can operate in regular or irregular clocking modes and can appropriately XOR its contents with a value received from polynomial evaluation performed on more significant bits. This means that the core can also be used as a pseudo-random number generator, by initializing the LFSR lengths, polynomials, and clocking bits.

The core is connected to the ARM core via a 32-bit coprocessor interface. It is the responsibility of the software on the ARM core to appropriately load and use the two 114-bit keystream pairs. In addition, the module has a 3-bit address port and a read/write strobe signal interface with the coprocessor. Once a keystream has been generated, the plaintext encryption can be done outside the core.

The A5/1 core is initialized by writing to Coprocessor 15 register CR6. Keystream data is obtained from the core by reading from Coprocessor 15 register CR8. These registers use self-incrementing counters, so data must always be written to or read from them in groups of eight words. The initialization data sequence is presented in Table 2.3, and the keystream data sequence is presented in Table 2.4.

Table 2.3: Initialization Sequence: Coprocessor 15 Register CR6

Index	Bits	Description
0	[7:0]	LFSR 0 length
0	[15:8]	LFSR 1 length
0	[23:16]	LFSR 2 length
0	[31:24]	Reserved
1	[31:0]	LFSR 0 polynomial
2	[31:0]	LFSR 1 polynomial
3	[31:0]	LFSR 2 polynomial
4	[3:0]	LFSR 0 clocking bit
4	[7:4]	LFSR 1 clocking bit
4	[11:8]	LFSR 2 clocking bit
4	[31:12]	Reserved
5	[31:0]	LFSR 0 session key
6	[31:0]	LFSR 1 session key
7	[21:0]	LFSR 2 session key

Table 2.4: Keystream Sequence: Coprocessor 15 Register CR8

Index	Description
0	Keystream 0 bits [31:0]
1	Keystream 0 bits [63:32]
2	Keystream 0 bits [95:64]
3	Keystream 0 bits [127:96]
4	Keystream 1 bits [31:0]
5	Keystream 1 bits [63:32]
6	Keystream 1 bits [95:64]
7	Keystream 1 bits [127:96]



# 3 | Smart Memory Subsystem Errata

## 3.1 Errata

Erratum 762: The address pin count for the system and for the Memory Controller subsystem is 28 instead of 24.

## 3.2 I/O Description

Erratum 762: The following I/O signal width was corrected:

Table 3.1: Subsystem I/O Signals (Corrected)

Signal	In/Out	Width	Description
MEM_ADDR	Out	28	Off-chip memory address. Provides the base address (or the start address in case of a burst) of the data to be accessed.

## 4 | Maintenance Subsystem Errata

### 4.1 Errata

Erratum 597: Support for runtime reconfiguration of the AXI4 mesh interconnect.

Erratum 600: A collection of subsystem run-time performance monitors was inserted into the test article.

### 4.2 Features

- Erratum 597: Runtime control of system interconnect routing algorithms

### 4.3 Block Diagram

Errata 597 and 600: The performance monitor infrastructure was incorporated into the Maintenance subsystem, and the Mesh Router control signals were connected to it.

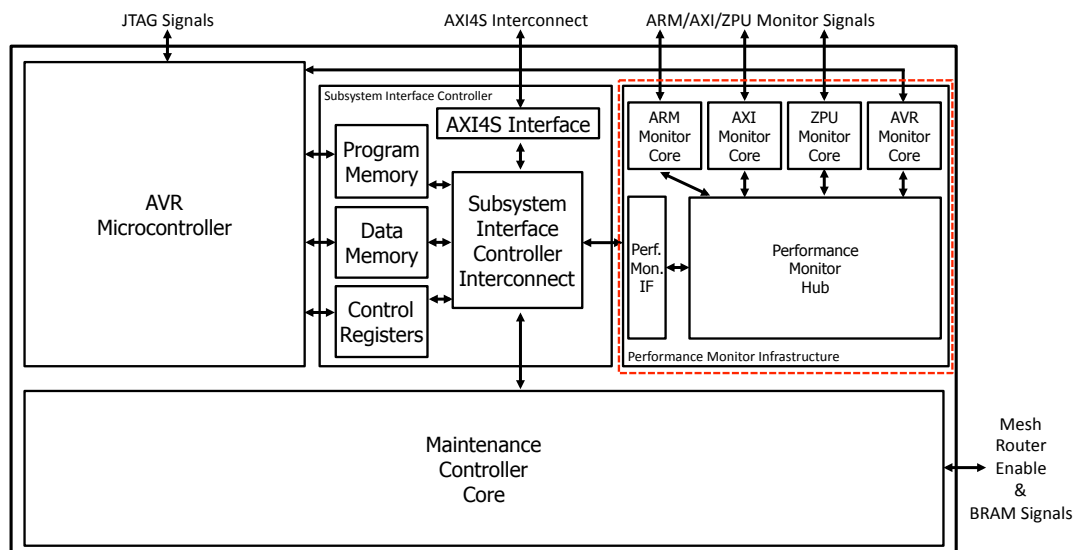


Figure 4.1: Maintenance Subsystem Block Diagram

### 4.4 I/O Description

Errata 597 and 600: The following I/O signals were added to the Maintenance Subsystem.

Table 4.1: Subsystem I/O Signals

Signal	In/Out	Width	Description
BRAM_ADDR	Out	10	maintenance subsystem control address
BRAM_DOUT	Out	8	maintenance subsystem control data
BRAM_WE	Out	9	maintenance subsystem control write enable

Table 4.1: Subsystem I/O Signals

Signal	In/Out	Width	Description
arm_cpuwait	In	1	ARM processor stall signal
zpu_cpuwait	In	1	ZPU processor stall signal
zpu_pc	In	32	ZPU processor program counter
zpu_inst	In	32	ZPU processor instruction register
axi_ports_empty	In	90	AXI4S interconnect Input FIFO empty signal
axi_ports_full	In	90	AXI4S interconnect Input FIFO full signal

## 4.5 Technical Details

Erratum 597: Run-time reconfiguration of the AXI4S mesh interconnect is controlled by the Maintenance subsystem through the use of the BRAM\_\* and ROUTER\_PAUSE input signals. The mesh tables are preloaded with an XY Dimension Order Routing algorithm. After pausing the routers, the Maintenance subsystem can use the BRAM\_\* signals to change the routing algorithm to any algorithm suitable for a 3-by-3 mesh network, such as YX Dimension Order Routing. Table 4.2 defines the addresses for each of the mesh routers.

Table 4.2: Maintenance Controller Core Address Map

Address Range	Router
0x24000000 – 0x24000FFF	Mesh Router 0
0x24010000 – 0x24010FFF	Mesh Router 1
0x24020000 – 0x24020FFF	Mesh Router 2
0x24030000 – 0x24030FFF	Mesh Router 3
0x24040000 – 0x24040FFF	Mesh Router 4
0x24050000 – 0x24050FFF	Mesh Router 5
0x24060000 – 0x24060FFF	Mesh Router 6
0x24070000 – 0x24070FFF	Mesh Router 7
0x24080000 – 0x24080FFF	Mesh Router 8

### 4.5.0.1 Module-Level Address Mapping

Erratum 597: The following address range is added to the Maintenance subsystem address map.

Table 4.3: Module-Level Address Mapping

Address Range	Core
0x24000000 – 0x24FFFFFFF	Maintenance Controller

### 4.5.1 Performance Monitors Infrastructure

Erratum 600: This entire subsection has been added as an erratum.

The performance monitor infrastructure provides run-time system information. The information can be collected and used by a designer to better understand the system performance under various loads and conditions. The system uses individual cores to monitor the ARM processor, ZPU processor, AVR processor, and AXI4S mesh interconnect routers. A designer can enable or disable monitoring and capture or reset each monitor core's data. The monitoring infrastructure is composed of the following blocks:

- Performance Monitor Interface
- Performance Monitor Hub
- ARM Performance Monitor Core
- AVR Performance Monitor Core
- ZPU Performance Monitor Core
- AXI4S Mesh Interconnect Monitor Core

The ARM subsystem, Cryptographic subsystem, and AXI4S Interconnect are separate from the Maintenance subsystem, but their monitoring cores reside within the Maintenance subsystem. Figure 4.1 shows the performance monitor infrastructure integrated into the Maintenance subsystem, including the subsystem I/O ports added for external monitoring of the ARM subsystem, Cryptographic subsystem, and AXI4S Interconnect.

#### 4.5.1.1 Performance Monitor Interface

The system interacts with the Performance Monitor through the Performance Monitor Interface. An additional port was added to Subsystem Interface Controller (SIC) interconnect. This port connects to the Performance Monitor Interface at address 0x24000000. The interface also adds separate 16-element deep FIFOs on the transmit and receive ports to buffer commands and data going to and from the system.

#### 4.5.1.2 Performance Monitor Hub

The Performance Monitor Hub aggregates commands from the system and passes them on to the specified performance monitor core. Table 4.4 defines the supported commands.

Table 4.4: Performance Monitor Commands

Command	Description
0x0	Retrieve all data from all performance monitors
0x1	Retrieve all data from a specific performance monitor
0x2	Retrieve a specific data word from all performance monitors
0x3	Retrieve a specific data word from one performance monitor
0x4	Reset data for all performance monitors
0x5	Reset data for a specific performance monitor
0x6	Enable data collection for all performance monitors
0x7	Enable data collection for a specific performance monitor
0x8	Disable data collection for all performance monitors
0x9	Disable data collection for a specific performance monitor

Table 4.5 enumerates the performance monitor cores. These numbers can be combined with commands to designate a specific performance monitor.

Table 4.5: Performance Monitor Cores Numeric Representation

Number	Core Name
0	AVR Processor Performance Monitor
1	ZPU Processor Performance Monitor
2	ARM Processor Performance Monitor
3	AXI4S Interconnect Performance Monitor

Table 4.6 describes the Performance Monitor Hub Command Register at address 0x24000000.

Table 4.6: Performance Monitor Hub Command Register

Bit number	Access	Description
[31:12]	—	Reserved
[11:8]	w	Monitor number (Table 4.5)
[7:4]	—	Reserved
[3:0]	w	Command (Table 4.4)

After a command is issued, the resulting data can be read from address 0x24000000. The data returned depends on the command that was issued. The first word of data indicates how many monitors are included in the results. Then for each monitor, the number of data words, followed by the actual data words are returned. A simple C program with a double-nested loop can be used to iterate over each monitor and then over each datum.

#### 4.5.1.3 AVR Performance Monitor Core

The AVR performance monitor core receives inputs `avr_cpuwait`, `avr_pc`, and `avr_inst`. When the `avr_cpuwait` signal is high, the AVR processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `avr_cpuwait` signal is active. Combined with the total run-time of the AVR subsystem, a user can quickly understand the utilization of the processor core. The monitor can also capture the current value of the program counter and the current instruction that is being executed. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AVR monitor includes two 64-bit timers and two 32-bit words for the program counter and current instruction. Timer 0 measures the idle time when the `avr_cpuwait` signal is asserted. Timer 1 measures the active run time, when the `avr_cpuwait` signal is not asserted. The monitor data is described in Table 4.7.

Table 4.7: AVR Performance Monitor Core Data

Word	Description
0	Timer 0: AVR processor idle timer [31:0] data
1	Timer 0: AVR processor idle timer [63:32] data
2	Timer 1: AVR processor run timer [31:0] data
3	Timer 1: AVR processor run timer [63:32] data
4	AVR processor program counter
5	AVR processor instruction register

#### 4.5.1.4 ZPU Performance Monitor Core

The ZPU performance monitor core receives inputs `zpu_cpuwait`, `zpu_pc`, and `zpu_inst`. When the `zpu_cpuwait` signal is high, the ZPU processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `zpu_cpuwait` signal is active. Combined with the total run-time of the ZPU subsystem, a user can quickly understand the utilization of the processor core. The monitor can also capture the current value of the program counter and the current instruction that is being executed. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ZPU monitor includes two 64-bit timers and two 32-bit words for the program counter and current instruction. Timer 0 measures the idle time when the `zpu_cpuwait` signal is asserted. Timer 1 measures the active run time, when the `zpu_cpuwait` signal is not asserted. The monitor data is described in Table 4.8.

Table 4.8: ZPU Performance Monitor Core Data

Word	Description
0	Timer 0: ZPU processor idle timer [31:0] data
1	Timer 0: ZPU processor idle timer [63:32] data
2	Timer 1: ZPU processor run timer [31:0] data
3	Timer 1: ZPU processor run timer [63:32] data
4	ZPU processor program counter
5	ZPU processor instruction register

#### 4.5.1.5 ARM Performance Monitor Core

The ARM performance monitor core receives input signal `arm_cpuwait`. When the `arm_cpuwait` signal is high, the ARM processor is stalled and waiting for data. When enabled, the monitor core counts the number of clock cycles the `arm_cpuwait` signal is active. Combined with the total run-time of the ARM subsystem, a user can quickly understand the utilization of the processor core. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The ARM monitor includes two 64-bit timers. Timer 0 measures the idle time when `arm_cpuwait` is asserted. Timer 1 measures the active run time, when `arm_cpuwait` is not asserted. The monitor data is described in Table 4.9.

Table 4.9: ARM Performance Monitor Core Data

Word	Description
0	Timer 0: ARM processor idle timer [31:0] data
1	Timer 0: ARM processor idle timer [63:32] data
2	Timer 1: ARM processor run timer [31:0] data
3	Timer 1: ARM processor run timer [63:32] data

#### 4.5.1.6 AXI4S Interconnect Performance Monitor Core

The AXI4S interconnect performance monitor core receives bus inputs `axi_port_empty` and `axi_port_full`. These signals reflect the AXI4S interconnect input FIFO status for the mesh network routers. The interconnect ports in the TA3B system have independent FIFOs to buffer incoming data. The FIFO status is useful for understanding the utilization of the interconnect and the load distribution of an application on the system. The performance monitor infrastructure allows this information to be collected at run-time and to be enabled, disabled, or reset at the user's discretion.

The AXI4S monitor data includes six 32-bit words indicating the status of the input FIFOs. The AXI4S interconnect has 9 ports. Each port has five FIFOs for incoming data, one in each Cartesian direction—North, South, East, West—and one for the subsystem's local port. This corresponds to 45 FIFO Empty signals and 45 FIFO Full signals, as shown in Figure 4.10.

Table 4.10: AXI4S Performance Monitor Core Data

Index	Bits	Description
0	[31:0]	FIFO empty signals
1	[45:32]	FIFO empty signals
1	[63:46]	Reserved
2	[95:64]	Reserved
3	[31:0]	FIFO full signals
4	[45:32]	FIFO full signals
4	[63:46]	Reserved
5	[95:64]	Reserved

The 45 used bits of each FIFO signal are ordered as follows as shown in Figure 4.11, where L, N, S, E, and W correspond to Local, North, South, East, and West port connections.

Table 4.11: AXI4S Performance Monitor Core Data

Index	Description	Subsystem
[4:0]	Router 0 signals (L, N, S, E, W)	Smart Memory subsystem
[9:5]	Router 1 signals (L, N, S, E, W)	ARM subsystem
[14:10]	Router 2 signals (L, N, S, E, W)	Maintenance subsystem
[19:15]	Router 3 signals (L, N, S, E, W)	Peripheral subsystem
[24:20]	Router 4 signals (L, N, S, E, W)	Cryptographic subsystem
[29:25]	Router 5 signals (L, N, S, E, W)	Hardware Control subsystem
[34:30]	Router 6 signals (L, N, S, E, W)	AVR Subsystem
[39:35]	Router 7 signals (L, N, S, E, W)	On-Chip Memory subsystem
[44:40]	Router 8 signals (L, N, S, E, W)	JTAG subsystem

# 5 | Peripheral Subsystem Errata

## 5.1 Errata

Erratum 602: Writable UART counters were inserted into the test article to allow runtime baud rate adjustments.

## 5.2 Technical Details

Erratum 602: The UART supports operations to receive and transmit data, to get and set the baud rate, to get the FIFO status, and to acquire, check, or release a mutex. The operation requested is determined by the read or write address from Table 5.1.

Table 5.1: UART Address Summary

Address	Description
0x32000000	Normal Operation
0x32000004	Get/Set Baud Low
0x32000008	Get/Set Baud High
0x3200000C	Get FIFO Status
0x32000010	Check Mutex
0x32000010	Acquire Mutex
0x32000020	Release Mutex

Erratum 602: The UART baud rate is controlled by two 32-bit registers. The low 12 bits at address 0x32000004 set the baud frequency and the low 16 bits at address 0x32000008 set the baud limit. These registers together set two internal counters that configure the baud clock.

Erratum 602: The UART default baud rate is 57,600 bps. Table 5.2 shows the baud rate settings to use if the system clock frequency is 50 MHz.



Table 5.2: UART Settings

Baud Rate	baud_freq	baud_limit
300	0x0003	0xF421
600	0x0003	0x7A0F
1,200	0x0003	0x3D06
2,400	0x0006	0x3D03
4,800	0x000C	0x3CFD
9,600	0x0018	0x3CF1
14,400	0x0024	0x3CE5
19,200	0x0030	0x3CD9
28,800	0x0048	0x3CC1
38,400	0x0060	0x3CA9
56,000	0x001C	0x0C19
<b>57,600<sup>†</sup></b>	<b>0x0090</b>	<b>0x3C79</b>
115,200	0x0120	0x3BE9
128,000	0x0040	0x0BF5
153,600	0x0180	0x3B89
230,400	0x0240	0x3AC9
256,000	0x0080	0x0BB5
460,800	0x0480	0x3889
921,600	0x0900	0x3409
1,382,400	0x0D80	0x2F89
2,304,000	0x0480	0x07B5
4,608,000	0x0900	0x0335

<sup>†</sup> Default baud rate

Erratum 602: The baud settings in Table 5.2 can be calculated from the desired baud rate as follows:

$$Baud\_freq = \frac{16 \times baud\_rate}{gcd(system\_clock\_freq, 16 \times baud\_rate)}$$

$$Baud\_limit = \frac{system\_clock\_freq}{gcd(system\_clock\_freq, 16 \times baud\_rate)} - baud\_freq$$

## 6 | Cryptographic Subsystem Errata

### 6.1 Errata

Erratum 599: The ZPU processor's data memory is connected to the JTAG chain.

Erratum 600: A collection of subsystem run-time performance monitors was inserted into the test article.

Erratum 686: The SHA-3 hash function can be bypassed.

Erratum 753: The SHA-3 Skein candidate was added to the Cryptographic subsystem.

### 6.2 Features

Erratum 753: The following features were added to the Cryptographic subsystem.

- Efficient implementation of SHA-3 candidates Blake, JH, and Skein
- Runtime selection of three cryptographic cores

### 6.3 I/O Description

Erratum 600: The following I/O pins were added to the Cryptographic Subsystem.

Table 6.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
zpu_cpuwait	Out	1	ZPU processor stall signal
zpu_pc	Out	32	ZPU processor program counter
zpu_inst	Out	32	ZPU processor instruction register

### 6.4 Technical Details

#### 6.4.1 ZPU Core

Erratum 599: The ZPU processor data memory can be configured through the JTAG subsystem. This memory resides within the Subsystem Interface Controller (SIC). JTAG write transactions starting at base address 0x41000000 are written to the ZPU data memory, while write transactions starting at base address 0x42000000 are written to the SIC control registers.

Erratum 600: The ZPU processor's fetch stall signal, program counter, and current instruction register are connected to the Maintenance Subsystem. The processor stalls when it performs I/O transactions to memory. More details regarding the monitoring of the ZPU processor's cpuwait, program counter, and instruction register signal can be found in Chapter 4.

#### 6.4.2 Subsystem Interface Controller

##### 6.4.2.1 SIC Control Registers

Errata 686 and 753: When the select bits for the Cryptographic algorithm are set to 10, the Skein hash function is selected. When the bits are set to 11, the encryption is bypassed and the input plaintext is passed to the output.

Table 6.2: Remote Request Register (Corrected)

Bit number	Access	Description
[31:18]	—	Reserved (bit 17 is now part of the algorithm selection)
[17:16]	r/w	Crypto Core algorithm select (00 = JH, 01 = Blake, 10 = Skein, 11 = Bypass) [default:00]
[15:14]	r/w	Data memory read margin B adjust [default: 00]
[13:12]	r/w	Data memory read margin A adjust [default: 00]
[11:10]	r/w	Program memory read margin B adjust [default: 00]
[9:8]	r/w	Program memory read margin A adjust [default: 00]
[7:2]	—	Reserved
1	r/w	When 1, initiates remote data memory requests [default: 0]
0	r/w	When 1, initiates remote program memory requests [default: 0]

### 6.4.3 Crypto Core

Erratum 686: This subsystem provides three cryptographic cores: JH, Blake, and Skein. Before using the Crypto core to generate a hash for a message, the user must select the proper algorithm. Write 0x00000000 for JH, 0x00010000 for Blake, or 0x00020000 for Skein to the SIC Remote Request Register at address 0x42000010. JH is selected by default.

#### 6.4.3.1 Skein Implementation

Erratum 753: The Skein algorithm is based on the Threefish block cipher. It uses Unique Block Iteration to compress the block cipher for faster hardware and software performance. The primary proposal for Skein is SHA-512 with a 64-bit input word size and a 512-bit output. Documentation and details about the algorithm internals can be found on the author's website: <http://www.skein-hash.info>.

## 7 | Hardware Control Subsystem Errata

### 7.1 Errata

Erratum 598: Changed the mesh network port width for the Hardware Control kernel to be 16 bits wide rather than 32 bits wide.

### 7.2 Technical Details

Erratum 598: The Hardware Control subsystem uses a 16-bit data connection, and has additional circuitry for conversion to the AXI4S 32-bit width of the mesh network. This includes a four element deep FIFO which acts as an additional buffer between the 32- and 16-bit interfaces.

## 8 | AVR Subsystem Errata

### 8.1 Errata

No errata exist for this subsystem.

## 9 | On-Chip Memory Subsystem Errata

### 9.1 Errata

No errata exist for this subsystem.

## 10 | JTAG Subsystem Errata

### 10.1 Errata

Erratum 599: The ZPU processor's data memory is connected to the JTAG chain.

### 10.2 Technical Details

Erratum 599: The JTAG subsystem can be used to configure the ZPU processor data memory in the Cryptographic subsystem, as described in Chapter 6. The JTAG subsystem is used to aggregate JTAG commands for the AVR, ARM, and ZPU memories. JTAG write transactions starting at base address 0x41000000 will be written to the ZPU data memory. Write transactions starting at base address 0x42000000 will be written to the Cryptographic subsystem SIC control registers.

# 11 | AXI4 Mesh Interconnect Errata

## 11.1 Errata

Erratum 597: Supports runtime reconfiguration of the AXI4 mesh interconnect.

Erratum 598: Changed the mesh network port width for the Hardware Control kernel to be 16 bits wide rather than 32 bits wide.

Erratum 600: A collection of subsystem runtime performance monitors was inserted into the test article.

## 11.2 I/O Description

Erratum 600: The following I/O pins were added to the AXI4 Interconnect.

Table 11.1: Subsystem I/O Signals (Added)

Signal	In/Out	Width	Description
axi_ports_empty	Out	90	AXI Interconnect Input FIFO empty signal
axi_ports_full	Out	90	AXI Interconnect Input FIFO full signal

## 11.3 Technical Details

Erratum 597: Run-time reconfiguration of the AXI4S mesh interconnect is controlled by the Maintenance subsystem through the use of the BRAM\_\* and ROUTER\_PAUSE input signals. The mesh tables are preloaded with an XY Dimension Order Routing algorithm. After pausing the routers, the Maintenance subsystem can use the BRAM\_\* signals to change the routing algorithm to any algorithm suitable for a 3-by-3 mesh network, such as YX Dimension Order Routing. The reconfigurability of the routing table was alluded to in the release documentation, but none of the means or details from Chapter 4 were provided.

Erratum 598: The Hardware Control subsystem—Port 5 on the mesh network—uses a 16-bit data connection, and has additional circuitry for conversion to the AXI4S 32-bit data transaction width. This includes a four element deep FIFO which acts as an additional buffer between the 32- and 16-bit interfaces.

Erratum 600: All of the input port FIFO status signals in the AXI4X interconnect are connected to the performance monitors in the Maintenance Subsystem. Additional details regarding the FIFO monitoring is available in Chapter 4.



---

# **ITAG FPGA INDEPENDENT FUNCTIONAL TESTING REPORT**

Information Sciences Institute  
University of Southern California

Modified: July 11, 2015

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Specifications . . . . .	3
1.2	Overview . . . . .	3
1.3	Organization . . . . .	4
<b>2</b>	<b>Infrastructure</b>	<b>5</b>
2.1	Test Agent . . . . .	5
2.2	Testing . . . . .	5
2.3	Clocking . . . . .	5
<b>3</b>	<b>Logic Testing</b>	<b>7</b>
3.1	Organization . . . . .	7
3.2	Scripts . . . . .	7
3.3	Controller . . . . .	8
3.4	Generation . . . . .	9
3.4.1	Groups 1–3: Logic . . . . .	9
3.4.2	Group 4: Distributed RAM . . . . .	14
3.4.3	Group 5: Shift Registers . . . . .	16
3.4.4	Group 6: Vertical Carry Chains . . . . .	16
3.4.5	Coverage . . . . .	17
<b>4</b>	<b>Interconnect Testing</b>	<b>18</b>
4.1	Approach . . . . .	18
4.2	Generation . . . . .	18
4.3	Procedure . . . . .	19
4.4	Estimated Coverage . . . . .	19
<b>5</b>	<b>User's Guide</b>	<b>20</b>
5.1	System Requirements . . . . .	20
5.2	Test Generation . . . . .	20
5.3	Test Execution . . . . .	20
<b>6</b>	<b>Verification</b>	<b>21</b>
6.1	Logic Setting Coverage . . . . .	21
6.2	Interconnect Coverage . . . . .	21
6.3	Bitstream Coverage . . . . .	21
6.4	Operation . . . . .	22
<b>7</b>	<b>Coverage</b>	<b>23</b>
<b>8</b>	<b>Conclusion</b>	<b>24</b>
8.1	Revisions . . . . .	24
8.2	Future Work . . . . .	24
8.2.1	Other Logic Sites . . . . .	24
8.2.2	Additional Interconnect . . . . .	24
8.2.3	Fault Isolation and Fault Diagnostics . . . . .	24
8.2.4	Testing Time Reduction (PIP Packing) . . . . .	24
8.2.5	Testing Time Reduction (Test Order Optimization) . . . . .	24
8.2.6	Timing Verification . . . . .	24
8.2.7	Shorting Fault Model . . . . .	25
8.2.8	I/O Pin Testing . . . . .	25

# 1 | Introduction

This report describes Independent Functional Testing capabilities for Xilinx 7-Series FPGAs developed by USC Information Sciences Institute (ISI) and the Virginia Tech Configurable Computing Lab. The goal of this work is to test the majority of the functionality of a supported FPGA against stuck-at faults. Stuck-at faults are electrical faults in which signals are permanently stuck in the logic 1 or logic 0 state and are unable to change states.

There are various solutions to this problem presented in published literature, but none of them are comprehensive. This is the first solution we know of that includes independent functional testing as well as independent coverage metrics.

Our Statement of Work only requires that we test logic slices, so the interconnect test development that we performed is an added benefit. ISI developed the coverage assessment and verification code, while Virginia Tech developed the logic and interconnect testing approach and test generation.

## 1.1 Specifications

These tests support all four families within the Xilinx 7-Series: Virtex7, Kintex7, Artix7, and Zynq7000. The only devices not supported are those with more than one Super Logic Region (SLR): XC7VH580T, XC7VH870T, XC7VX1140T, and XC7V2000T.

Each test generates a PASS/FAIL response. The test coverage is sufficient to determine with a high level of confidence whether the Device Under Test (DUT) is genuine and operating correctly.

## 1.2 Overview

Modern FPGAs can contain tens of millions of configurable wires and hundreds of thousands of configurable logic sites. Testing this many resources raises a variety of technical challenges: FPGAs are portrayed as being highly regular and therefore excellent candidates for parallelism, but while that characterization is generally true, there are many nuances and exceptions at very low levels of abstraction.

Testing for stuck-at faults requires separately passing a logic 1 and a logic 0 through every covered path: every configurable interconnect resource and every configurable logic resource. This is accomplished with a “launch and capture” approach, where signals are launched from stateful elements along a path through reconfigurable resources, and are then captured by stateful elements. If both a logic 1 and a logic 0 can pass unaltered through each configurable resource, then none of the elements on that path can be permanently stuck at any particular logic state, and stuck-at faults along that path are disproved.

It is not possible to test all configurable paths in a single pass because nearly any selected path will block other paths. In the best case we can only test one set of non-conflicting resources in any single pass, and collect multiple sets of tests for use in multiple passes.

These tests focus on the most abundant resources in the device, specifically including SLICEL and SLICEM for the logic resources, and the INT tile wiring for interconnect resources.

The Zynq XC7Z020 contains a total of 24,240 logic sites of 88 different types. 13,300 of those are slices, 8,810 are power sources, and the remaining 2,130 are an assortment of DSPs, BRAMs, clock logic, high-speed transceivers, and other logic. By covering the slices and power sources, we achieve 91 % coverage of logic sites in this device. The percent coverage increases for larger devices, because they contain a larger percentage of slices.

### **1.3 Organization**

Chapter 2 begins by describing the testing infrastructure, assumptions, and requirements. Chapters 3 and 4 then presents the logic testing and interconnect testing, respectively. Chapter 5 provides a user's guide for test generation and execution. Chapter 6 discusses verification of logic, interconnect, and bitstream coverage, which is then quantified in Chapter 7. And Chapter 8 presents concluding remarks and discusses future work.

## 2 | Infrastructure

Our in-circuit testing approach assumes that the FPGA Device Under Test (DUT) is mounted on a PCB, and that special test access to external FPGA I/O pins is not available. This precludes the use of clock, reset, control, and monitoring signals. Other testing efforts in published literature do not accommodate these same restrictions.

Required testing connectivity consists of power and an interface to the device Configuration Controller—either JTAG or SelectMAP.

### 2.1 Test Agent

A test agent is needed to upload the test bitstreams, execute the tests, and collect the test results. This can include any of the following: A host PC with a JTAG cable, an internal agent such as the ARM core on a Zynq device, or an external micro-controller connected to the JTAG or SelectMAP ports.

The test agent must have enough storage for thousands of full configuration bitstreams, typically tens of gigabytes, depending on the target device. The test agent must also provide an API to control the configuration port and support these functions:

- `bool DownloadBitstream(string filename):` Download specified bitstream and confirm that the bitstream is active. Support for partial bitstreams is not currently required.
- `word ReadStatusRegister(void):` Poll and return the state of DONE in the Configuration Controller STAT register.
- `WriteAXSSRegister(uint32 word):` Write a 32-bit word to the AXSS register.
- `Readback(void):` Read back part of all of the configuration bitstream. Not required at present but reserved for fault diagnosis in the future. Readback is not required to test the FPGA interconnect.

The test agent should be able to execute simple instructions using the aforementioned API and a for or while loop. Trivial bitwise operators are required but arithmetic operators are not.

### 2.2 Testing

An implicit assumption is made that the interconnect is good when the logic is being tested, and the logic is good when the interconnect is being tested. As long as both the logic and interconnect tests are executed, faults in either of these will be detectable.

Each device in the 7-Series families has its own unique tile map and consequently its own unique bitstream. This means that a separate test suite must be developed for each device from parameterized test constructors.

It is also necessary to read the result back from the FPGA after each test, but we cannot rely on user I/O to do so. A few alternatives are available, but we have chosen to use STARTUPE2 pin USRDONE0. We can selectively drive this pin onto the board with USRDONETS, but it is simpler to simply read back its value from the Configuration Controller STATUS register.

### 2.3 Clocking

The inability to rely on external I/O for testing requires some other clocking source to run the tests. At a minimum, all tests need clocked registers to capture results, and some tests also need a register to determine whether they are testing sa1 or sa0 faults.

7-Series FPGAs include a few internal clocking options, some of which fit our needs. The internal configuration clock available on STARTUPE2 pin CFGMCLK is documented in the 7 Series FPGAs Configuration User Guide (UG470). This 65 MHz clock can be driven onto the global clock network and serves our basic clocking requirements.

## 3 | Logic Testing

The testing infrastructure currently supports all SLICEL and SLICEM logic sites and most TIEOFF logic sites. This three site types represent the vast majority of all logic sites in the device.

The remaining site types in 7-Series devices are unsupported at present. Most of them pertain to clocks, FIFOs, gigabit transceivers, I/Os, PCIe, and BRAM:

AMS_ADC	DSP48E1	IDELAYE2.FINEDELAY	MTBF2	PLLE2.ADV
AMS_DAC	EFUSE_USR	ILOGICE2	ODELAYE2	PMV
BSCAN	FIFO18E1	ILOGICE3	ODELAYE2.FINEDELAY	PMV2
BSCAN_JTAG_MONE2	FIFO36E1	IN_FIFO	OLOGICE2	PMV2_SVT
BUFG	FRAME_ECC	IOB	OLOGICE3	PMVBRAM
BUFGCTRL	GCLK_TEST_BUF	IOB18	OPAD	PMVIOB
BUFG_LB	GLOBALSIG	IOB18M	OSERDESE2	PS7
BUFHCE	GTHE2.CHANNEL	IOB18S	OUT_FIFO	RAMB18E1
BUFIO	GTHE2.COMMON	IOB33	PCIE_2.1	RAMB36E1
BUFMRCE	GTPE2.CHANNEL	IOB33M	PCIE_3.0	RAMBFIFO36E1
BUFR	GTPE2.COMMON	IOB33S	PHASER_IN	STARTUP
CAPTURE	GTXE2.CHANNEL	IOBM	PHASER_IN_ADV	USR_ACCESS
CFG_IO_ACCESS	GTXE2.COMMON	IOBS	PHASER_IN_PHY	XADC
DCI	GTZE2.OCTAL	IOPAD	PHASER_OUT	
DCIRESET	IBUFDS_GTE2	IPAD	PHASER_OUT_ADV	
DNA_PORT	ICAP	ISERDESE2	PHASER_OUT_PHY	
DRP_AMS_ADC	IDELAYCTRL	KEY_CLEAR	PHASER_REF	
DRP_AMS_DAC	IDELAYE2	MMCME2.ADV	PHY_CONTROL	

### 3.1 Organization

Slice testing is divided into six groups. These group numbers have well-defined meanings to the build scripts:

1. LUTs
2. Combinational paths through AMUX / BMUX / CMUX / DMUX
3. Combinational paths through AFFMUX / BFFMUX / CFFMUX / DFFMUX
4. SelectRAM (distributed LUT RAM)
5. Shift registers
6. Carry chains

### 3.2 Scripts

Each group test is generated with the help of a controller, a set of logic cells, and a top-level design. The design instantiates the controller and connects the cells to each other. The design is generated by a C++ Torc application that accepts a pair of rectangular coordinates and a mode as inputs and creates a testgen\*.v file.

Table 3.1: Logic testing groups.

Group	Design	Cells	Directory	# Tests
1	testgen.v	slicel.v	lut	2
2	testgen.v	slicel.v	config_with_FF	8
3	testgen.v	slicel.v	FFs	7
4	testgen_s_ram.v	s_ram32.v, s_ram64.v, s_ram128.v, s_ram256.v	SelectRAM	4
5	testgen_shiftreg.v	srl16.v, srl32.v	shiftreg	2
6	testgen_carrychain.v	<i>none</i>	VCARRY	1

A collection of scripts uses the Xilinx tools to generate XDL for the target device, while additional scripts modify the XDL design for the current test. These scripts coordinate the generation of the test files and for each of the six groups:

`extract_dut.sh`: Extracts the various parts of the design, including instances and nets of the DUT and the controller.

`swap_outpin.sh <config>`: Modifies the nets and slice configurations for the DUT. For example, the Xilinx tools creates certain datapaths that go from LUT output O5 to AMUX, but this script can force that datapath through output O6 to AMUX.

`combine_xdl.sh`: Merges the modified extracted XDL parts into the new XDL design.

`config_lut.sh <config>`: Modifies the LUT equation in each DUT slice. The original HDL design includes a dummy LUT equation to prevent optimization, but this script modifies the LUT equation as needed.

`compile.sh <config>`: Compiles HDL files using the Xilinx tools, and invokes various scripts to generate `temp.bit`.

### 3.3 Controller

The testing process requires multiple configuration, each of which uses a small portion of the FPGA fabric for a controller to oversee the testing. The controller consists of a driver and a comparator, where the driver provides stimulus to the DUT, and the comparator observes the DUT output and compares it to the DUT input. The comparator result generates a PASS or FAIL signal on the FPGA's DONE pin, which can also be observed using readback.

For groups 1, 2, 3, and 6, the driver is implemented as a simple Finite State Machine (FSM). In odd numbered states, the driver switches the input vector that it applied to the DUT, and in even numbered states, the comparator compares the DUT input and output. If a mismatch is detected, the FAIL signal is latched and the DONE pin is driven high.

For Group 4, the controller tests the SelectRAM for memory faults with the MATS (Modified Algorithmic Test Sequence) test.

For Group 5, the controller tests shift-registers with two symmetric chains, and generates a FAIL result if the two do not match.

In general, the controller resides in one portion of the device, while the other portion is being tested. This is flipped in the complementary configuration when the controller and DUT positions are swapped for full fault coverage. More specifically, the presence of large gaps in the 7-Series fabric for ARM, PCIe, transceivers, and other cores makes it necessary to subdivide the device into contiguous rectangular blocks, where each of the blocks is tested in turn.



### 3.4 Generation

#### 3.4.1 Groups 1–3: Logic

The testing strategy for slices require two conditions to be met. (1) All paths within the slice are excited to 0 to detect stuck-at-1 (sa1) faults, and excited to 1 to detect stuck-at-0 (sa0) faults. (2) If a fault exists it must be propagated outside the FPGA to be observable. The first condition can be met with appropriate test vectors and design generation. The second condition is more difficult to meet.

The FPGA contains a large number of slices, and each slice has multiple output pins. Direct observation of these pins outside the FPGA is impossible because it would require on the order of 100,000 I/O pins. A more tractable approach is to chain the output of one slice to the input of the next slice, as depicted in Figure 3.1. Use of the identify function ensures correct propagation from slice outputs to subsequent slice LUT inputs, and the result of all the tests can be observed at the very end of the slice chain.

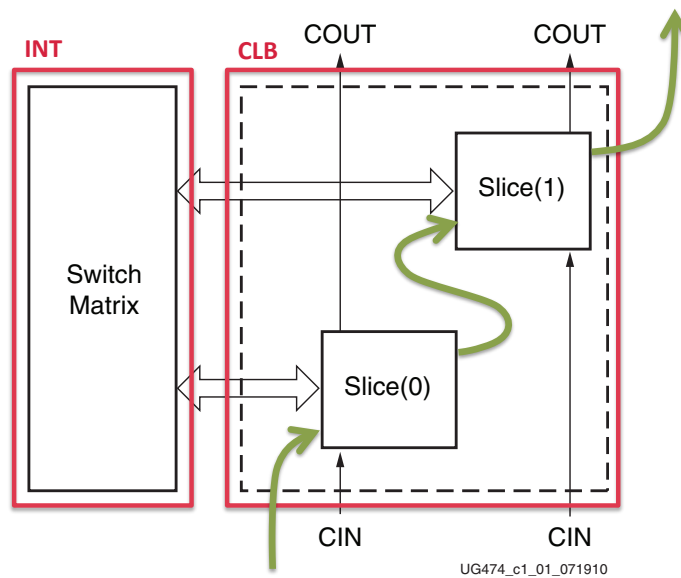
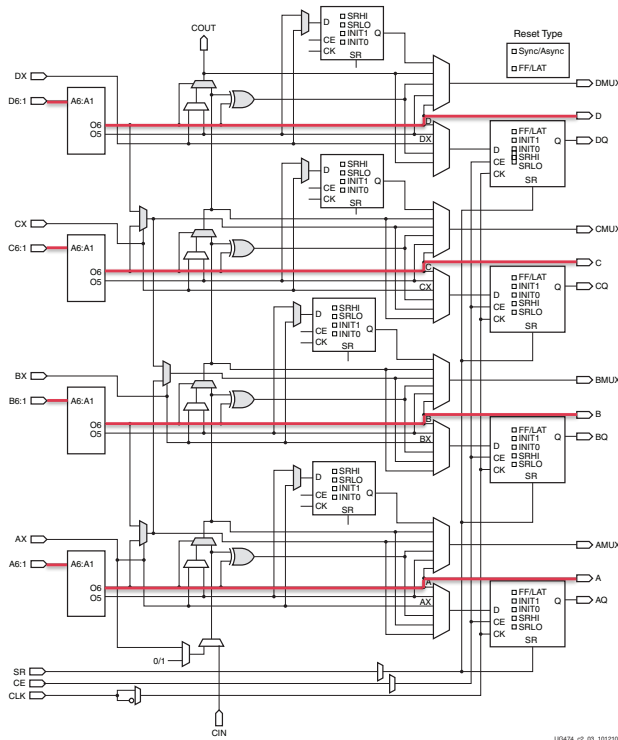
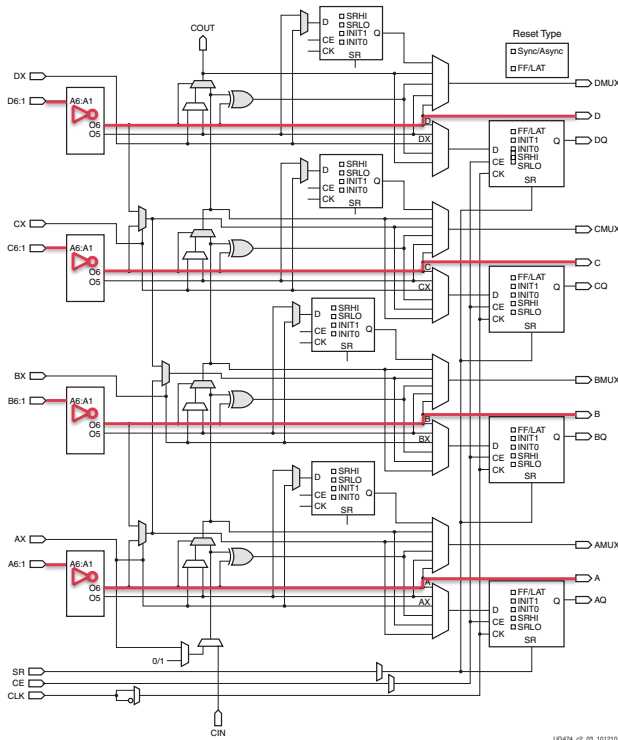


Figure 3.1: Output chaining from Slice(0) to Slice(1) to Slice(0) in next tile up.

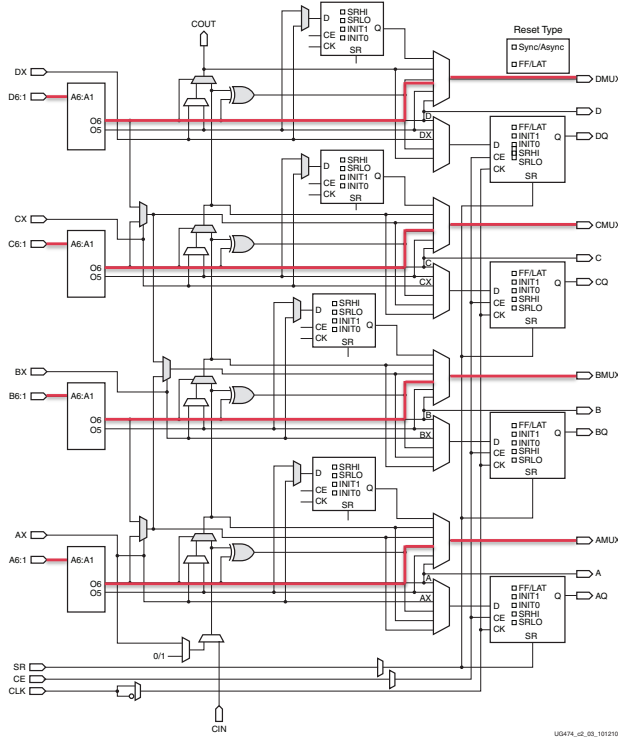
Table 3.2: SLICE Test Paths.



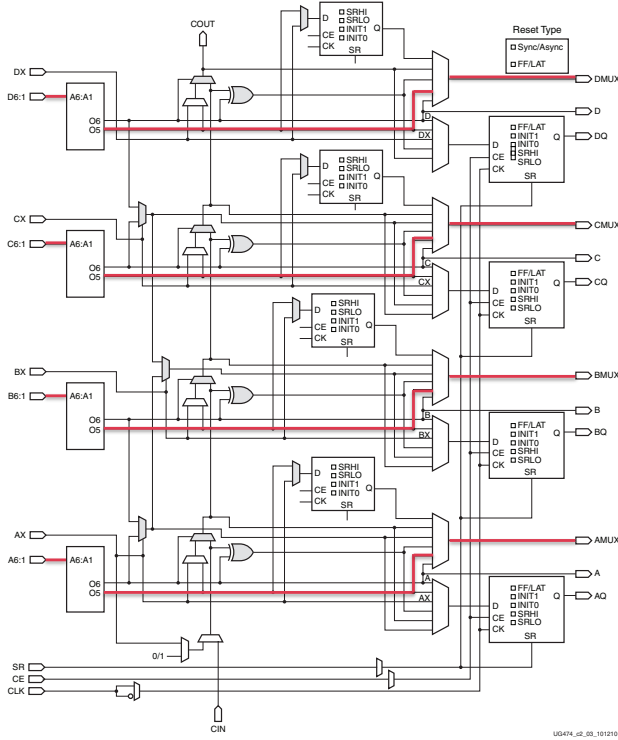
LUT: O6 to [ABCD]



LUT: O6 to [ABCD] (O6!=A1)



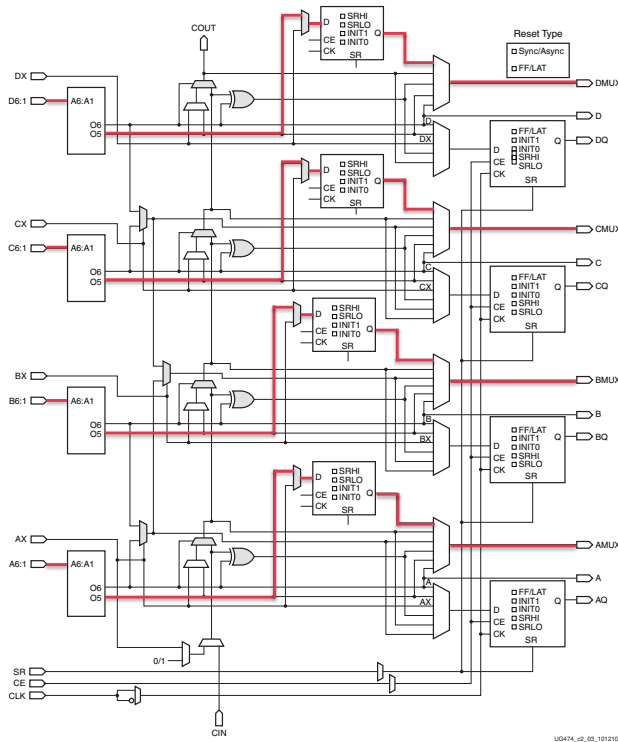
LUT: O6 TO [ABCD]MUX



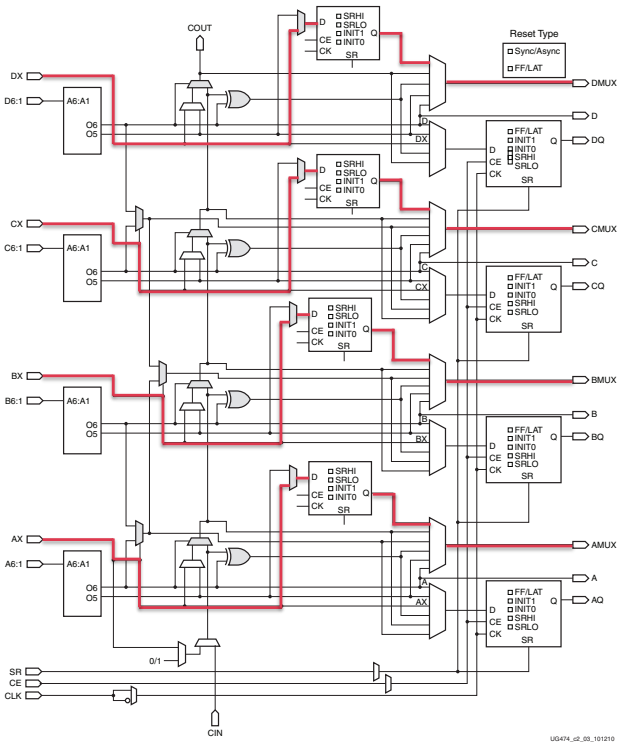
LUT: O5 TO [ABCD]MUX

11

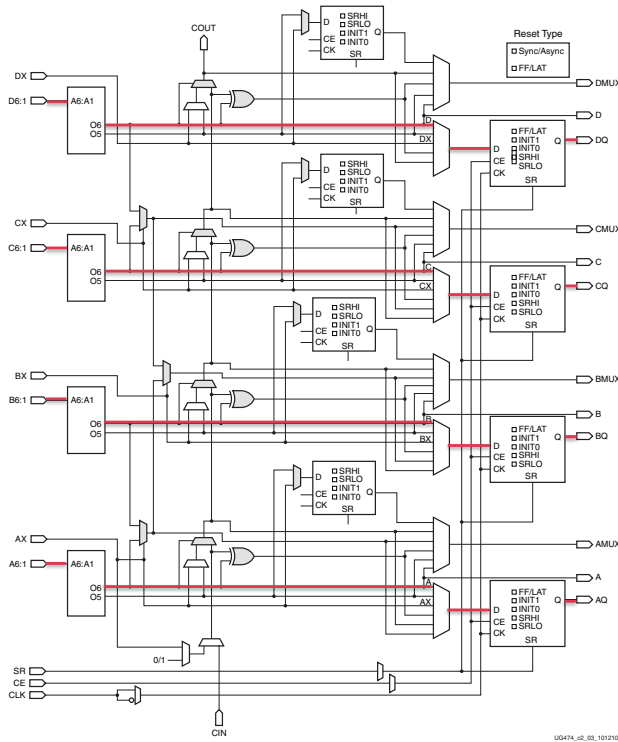
Table 3.2: SLICE Test Paths.



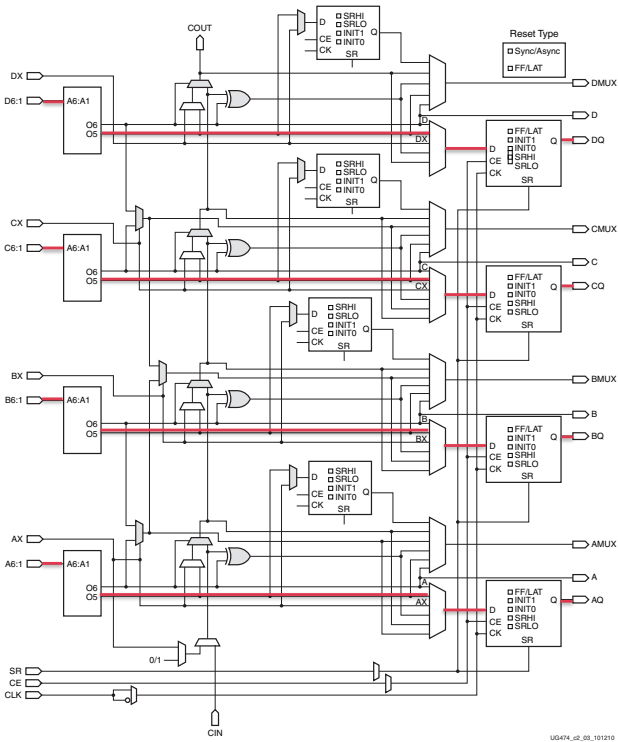
LUT: O5 through [ABCD]5FF to [ABCD]MUX



LUT: [ABCD]X through [ABCD]5FF to [ABCD]MUX



LUT: O6 through [ABCD]FF to [ABCD]Q

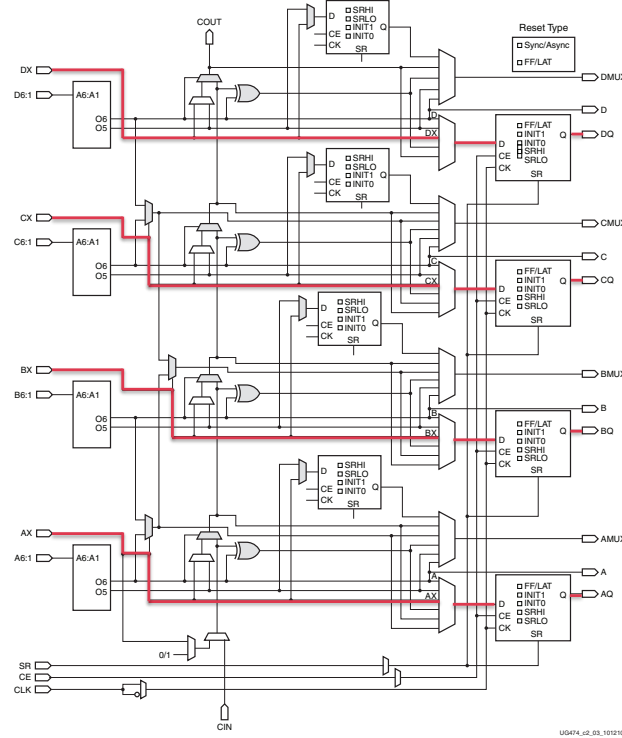


LUT: O5 through [ABCD]FF to [ABCD]Q

[illegible]

13

Table 3.2: SLICE Test Paths.



LUT: [ABCD]X through [ABCD]FF to [ABCD]Q

### 3.4.2 Group 4: Distributed RAM

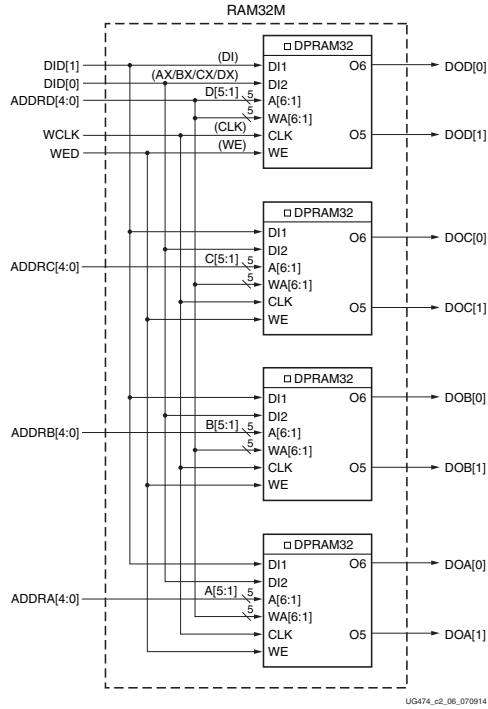
This group tests for faults in SLICEM SelectRAM with the MATS methodology. The widest address bus width required is 8 bits for the RAM256X1S mode. A simple 11-bit up/down counter is used to implement the MATS test, where the bits are interpreted as follows:

- counter[0]: write-enable signal, 0 to read, and 1 to write.
- counter[9:1]: if counter[0] = 0, march up, address[7:0] = counter[8:1]
- counter[9:1]: if counter[0] = 1, march down, address[7:0] =  $\sim$ counter[8:1]
- counter[10]: data bit to write

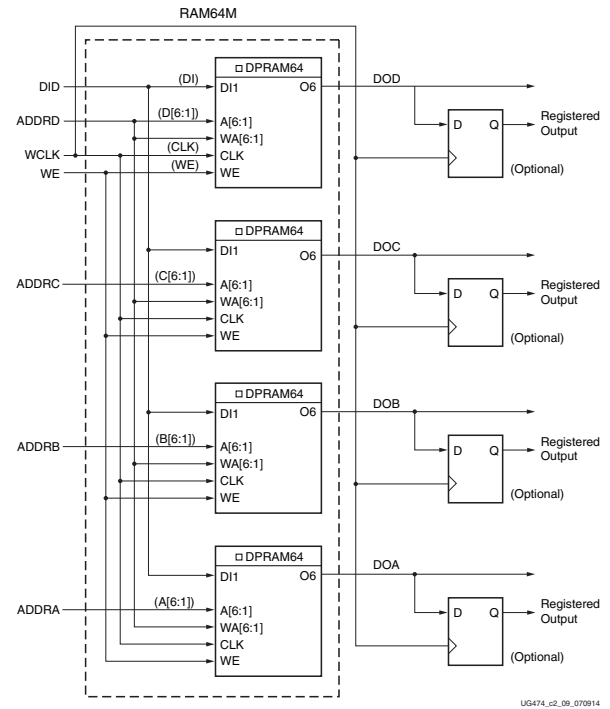
If the counter is incremented at rate CLK, then the memory is driven at rate CLK/2 on rising edges. This 11-bit free-running counter is sufficient to implement the basic MATS test.

- For counter values 000,0000,0000 to 001,1111,1111, memory is traversed in *increasing* order, with data value 0 written and read back on consecutive cycles.
- For counter values 010,0000,0000 to 011,1111,1111, memory is traversed in *decreasing* order, with data value 0 written and read back on consecutive cycles.
- For counter values 100,0000,0000 to 101,1111,1111, memory is traversed in *increasing* order, with data value 0 written and read back on consecutive cycles.
- For counter values 110,0000,0000 to 111,1111,1111, memory is traversed in *decreasing* order, with data value 0 written and read back on consecutive cycles.

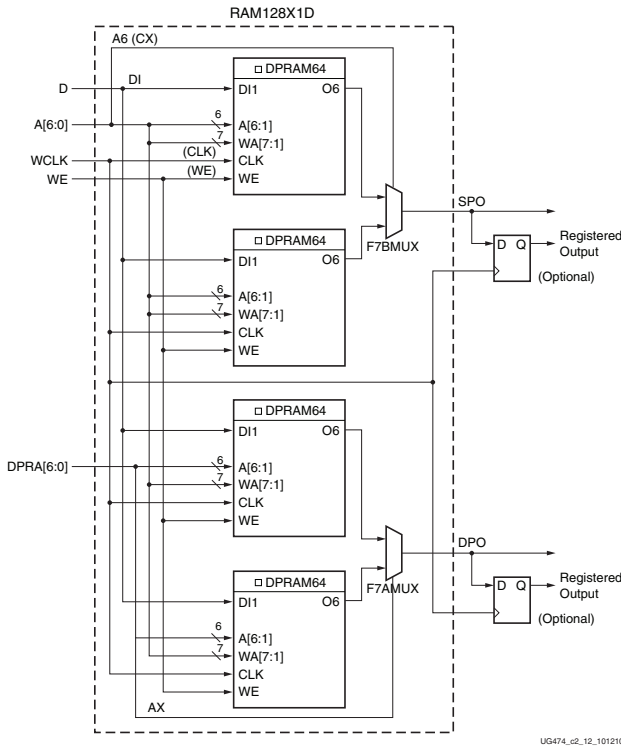
Table 3.3: SLICEM Memory Testing.



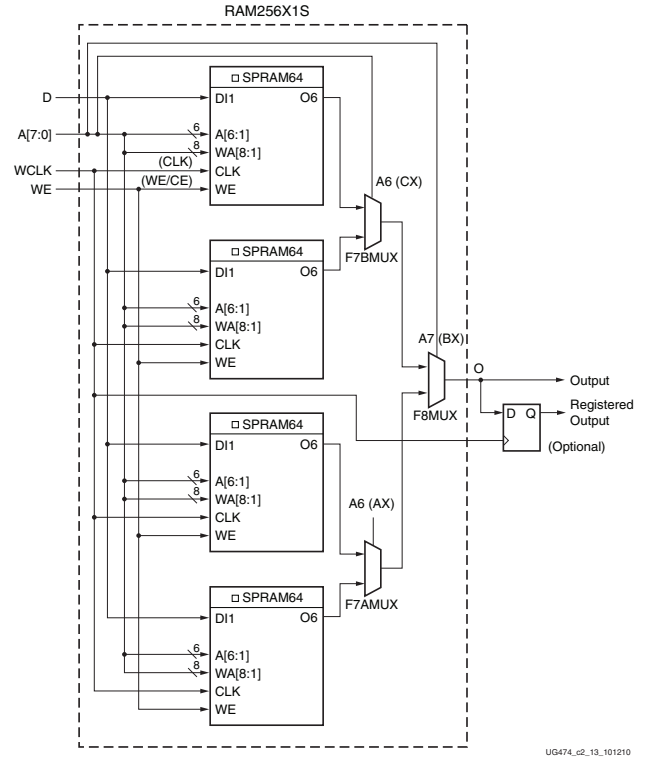
32 x 2 quad port distributed RAM



64 x 1 quad port distributed RAM



128 x 1 dual port distributed RAM



256 x 1 single port distributed RAM

### 3.4.3 Group 5: Shift Registers

Shift registers are tested for faults with a pair of long circular chains of equal length. The chains are initialize with an alternating 10 pattern, and the low-order bits of the two chains are XORed together, as depicted in Figure 3.2.

One configuration tests chains of SRL16 primitives, while the other configuration tests chains of SRL32 primitives. This allows both possible shift register modes to be tested for both sa0 and sa1 faults.

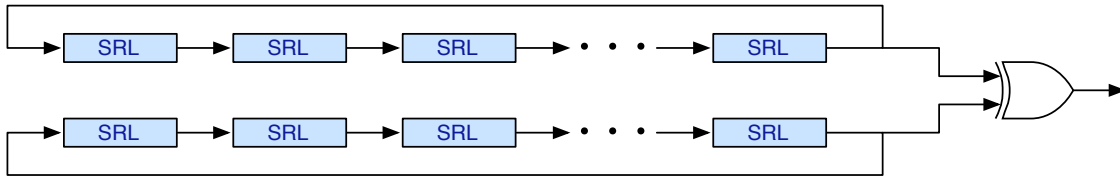


Figure 3.2: Pair of long circular SRL chains, with outputs XORed together for final result.

### 3.4.4 Group 6: Vertical Carry Chains

All available slices in each column are cascaded with COUT of one slice driving CIN of the next slice to form a chain. The output of the top slice is passed through DMUX and connected to the bottom slice CIN of the next column. This forms long vertical carry chains that span multiple columns. CIN is alternately excited with 1 and 0 to test both sa0 and sa1 faults.

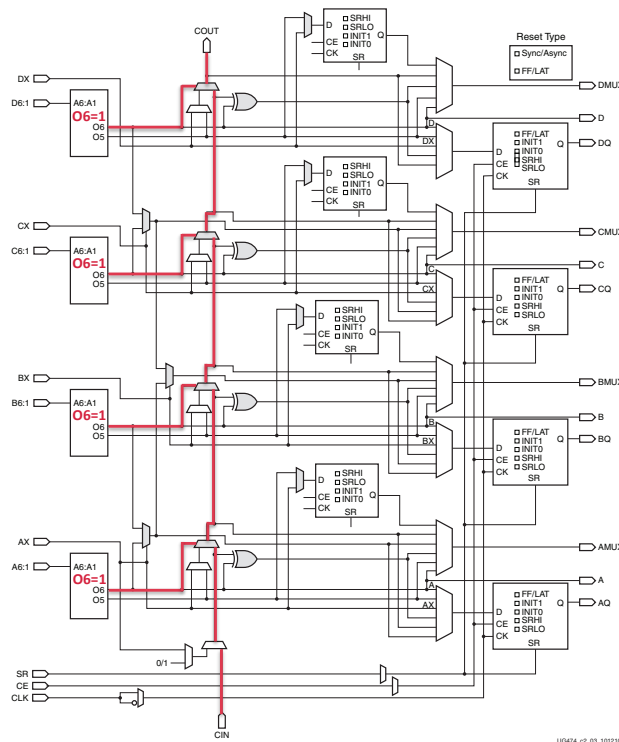


Figure 3.3: Carry propagation within a single slice.



### 3.4.5 Coverage

Each slice logic element and input/output port is covered by one or more test groups and configurations. The elements and ports are depicted in Figure 3.4, and the associated coverage of those ports and elements is shown in Table 3.4.

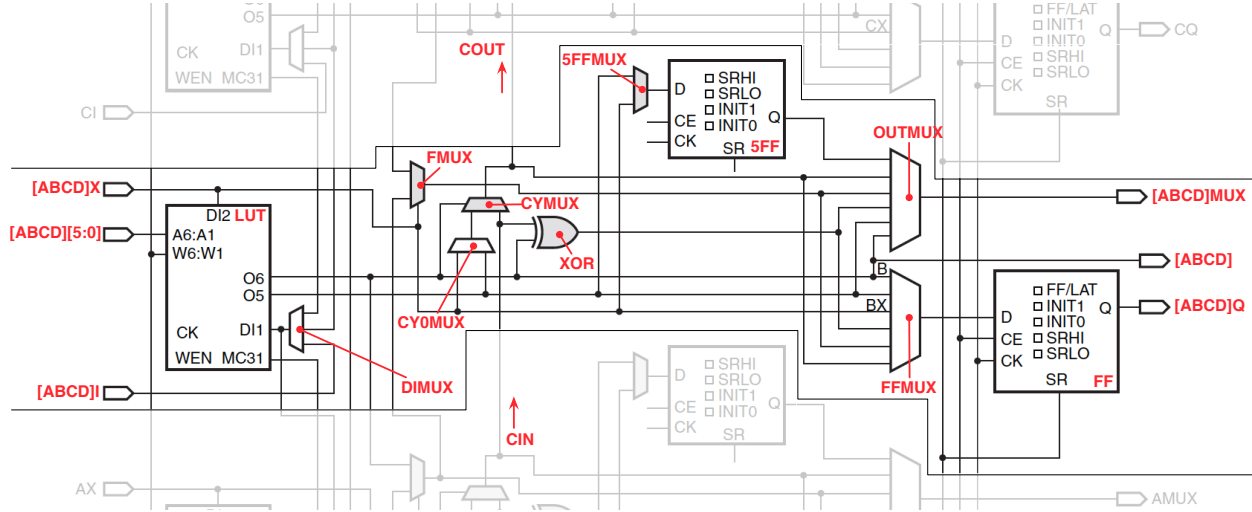


Figure 3.4: Covered SLICEM elements.

Table 3.4: SLICEM fault coverage by node.

Element	Group : Configuration
LUT	1:1, 1:2
OUTMUX	2:1, 2:2, 2:3, 2:4, 2:5, 2:6, 2:7, 2:8
FFMUX	3:1, 3:2, 3:3, 3:4, 3:5, 3:6, 3:8
XOR	2:5, 3:5
CY0MUX	2:3, 2:5
CYMUX	2:3, 2:4, 2:6
5FFMUX	2:7, 2:8
5FF	2:7, 2:8
FF	3:1
FMUX	4:2, 4:3, 4:4
LUT address decode	4:1, 4:2, 4:3, 4:4
LUT asynchronous read	4:1, 4:2, 4:3, 4:4
LUT SRL	5:1, 5:2
Input / Output	Group : Configuration
[ABCD]MUX	2:1
[ABCD]Q	3:1
[ABCD][5:0]	2:1
[ABCD]X	2:5
[ABCD]	1:1, 1:2
CIN	6:1
COUT	6:1

## 4 | Interconnect Testing

### 4.1 Approach

FPGAs consist of islands of irregular interconnect in a sea of regular interconnect. The 7-Series architecture pairs one INT interconnect tile with each CLB, BRAM, DSP, or transceiver tile. Testing the interconnect requires the ability to launch signals into intended parts of the interconnect, and to subsequently capture them. This work focuses on testing INT tiles that are paired with adjacent CLB tiles, and which account for the vast majority of all interconnect in the device.

The 7-Series interconnect is structured as a collection of wires linked by Programmable Interconnect Points (PIPs). User designs cannot create or modify wires in any way—they can only turn predetermined connections between wires on or off.

The smallest devices contain millions of wires and more than ten times that many PIPs. The interconnect architecture is such that 100 % wire coverage would not translate into 100 % PIP coverage, but conversely that 100 % PIP coverage would virtually guarantee 100 % wire coverage. The only missing wires would be certain permanently on-connections between logic sites. We consequently aim for high PIP coverage with the understanding that high wire coverage will follow as a consequence.

The 7-Series INT tile type defines 3,744 PIPs. In practice certain boundary conditions may reduce that number by a handful. Our approach is to visit each of these INT PIPs in turn and to simultaneously test the PIP in nearly every INT tiles at once. This approach ensures that testing time remains constant as the device size increases.

When testing INT tiles paired with adjacent CLB tiles, the result of each PIP test path is fed into a carry chain mux and propagated vertically. If the mux inputs are complementary and the select line is driven by a PIP test path result, then the propagated value will signal whether or not a fault occurred in the test. This setup is used for fault detection, but it also permits fault diagnosis through readback, where each register value indicates whether the associated PIP test path exhibited a fault.

### 4.2 Generation

Test generation is a five-step process:

1. Generate XDLRC architecture information for the target device. The two special environment variables `XIL_TEST_ARCS` and `XIL_DRM_EXCLUDE_ARCS` must be set to 1.
2. Preprocess the XDLRC to generate a device database for later use.
3. Use Synplify to synthesize the SLICE1 and SLICE2 modules into EDIF. XST is unable to synthesize these designs properly and causes a fatal error during the Xilinx `map` stage.
4. Generate and implement two or more designs that divide the device into separate DUT and controller regions.
5. Customize the generated XDL for each testable PIP in the INT tiles.

Most of these steps must be executed for each device to be supported. The XDL customization according to PIPs must be executed for every PIP in the INT tiles, which is  $O(3,700)$ . Devices with the same part number that differ only in packaging do not need to be treated separately.

A bash script—`generate.sh`—automates the generation process. It takes the design part number as a parameter and generates a directory containing the resulting configuration bitstreams. Even on a relatively fast machine, this step typically takes multiple days to execute, with most of the time taken by `xdl -xdl2ncd` and by `bitgen`. This process could be greatly accelerated in the future with the help of Torc Micro-Bitstreams and Virginia Tech's tFlow.

A relatively fast Core i54670K CPU running at 3.4 GHz with 32 GB of memory can normally generate a full XC7Z020 bitstream in 600 s of wall clock time, with most of the time consumed by DRC checks. Bitstream generation time can be reduced from 3,480 s to 357 s for the XC7Z0100 by passing a `-d` flag to bitgen. Further reduction is possible in `xdl` conversion by disabling DRC checks with the `-nodrc` flag, dropping from 4,080 s to 140 s

### 4.3 Procedure

The testing procedure is described by the following pseudo-code, and makes use of the API in Chapter 2.1:

```
For each test configuration bitstream:
    DownloadBitstream(filename)
    Wait(10ms)
    status = ReadStatusRegister()
    if (bit 14 of status is set):
        Report test failure for this test in phase 0
    Write AXSSRegister(1)
    Wait(10ms)
    status = ReadStatusRegister()
    if (bit 14 of status is cleared):
        Report test failure for this test in phase 1
If none of the tests reported failure:
    Report test pass
```

A bash script—`run.sh`—is provided to administer the test based on a directory of generated test configurations. A problem with the Xilinx `iMPACT` utility interferes with the testing process, so `iMPACT` is used only to read the device status register, and we recommend the open-source `xc3sprog` for bitstream configuration. Despite the name, `xc3sprog` works for a wide range of Xilinx architectures.

Testing time depends upon the device size, the number of test configurations, and the download speed. For the XC7Z020, each configuration takes about 20 s to download over a 6 MHz JTAG connection, so the full test for a device would complete within one day. When it is possible to use SelectMAP instead of JTAG, the download time can be reduced by a factor of 10.

### 4.4 Estimated Coverage

The predicted coverage for XC7Z020 INT tiles was 84.0 % of wires and 89.9 % of PIPs. These numbers are now known to be incorrect because of the many PIPs that are not being properly routed. Further discussion is provided in Chapter 7.

## 5 | User's Guide

### 5.1 System Requirements

- Xilinx ISE version 14.7 is required to properly support Zynq devices.
- The open-source Go language (<http://golang.org>) is required for XDLRC parsing, and for module and template test generation.
- The open-source xc3sprog (<http://xc3sprog.sourceforge.net>) programming utility is required for device configuration.

### 5.2 Test Generation

To begin test generation, set environment variable `BIST_PART` to the proper device designator, and remove stack size limits.

Bash:

```
#!/bin/bash
export BIST_PART=xc7k160tfbg676-1
ulimit -s unlimited
```

Csh:

```
#!/bin/tcsh
setenv BIST_PART xc7k160tfbg676-1
limit stacksize unlimited
```

After setting the `BIST_PART` environment variable, invoke `generate_all.sh`. This script will in turn invoke the `generate_all.sh` scripts inside each of the `ise` directories described in Table 3.1.

```
./generate_all.sh
```

It is worth noting that test generation is highly scripted and takes a very long time to complete.

### 5.3 Test Execution

The `$BIST/config_all.sh` script locates all generated bitstreams in subfolders of `$BIST` and appends their names to `$BIST/list`. Each bitstream in turn is uploaded to the FPGA and executed, after which the status of the `DONE` pin is read and logged into `$BIST/result`.

```
./config_all.sh
```

## 6 | Verification

The coverage verification utility provides an independent assessment of resources covered by the tests. The assessments are grouped into three categories: Site logic, interconnect, and bits.

In each case, the utility begins by creating a comprehensive list of resources that exist in the device. The utility then processes each XDL netlist or bitstream, and subtracts resources that it encounters from the larger list.

To guarantee the integrity of the effort, there was no overlap between the team developing the test suites and the team developing the coverage verification.

### 6.1 Logic Setting Coverage

Most logic sites contain anywhere from one to hundreds of configurable settings, and each of these can take on different values. The XDLRC data enumerates allowable values in most cases, but in a few other cases such as LUT masks, integer constants or bit patterns are used instead.

The setting coverage code accumulates every value for every setting in every logic site definition, and constructs a comprehensive list of setting values for the device. Every one of these value is internally flagged as unused, and the total number of setting values is noted. The flags are stored in a bit set for maximum efficiency.

As the logic coverage code visits every XDL instance in the set of test designs, for every setting value that is used in a design, the unused flag is cleared. The setting values still marked unused after inspection of each test suite are reported to the user. The final percent coverage is 100 % minus the number of uncovered setting values divided by the total number of setting values in the device.

### 6.2 Interconnect Coverage

The interconnect coverage tracking begins by inspecting every wire in the device, eliminating any pruned wires, and adding all remaining wires to a list. Each remaining wire is then expanded in turn to obtain a list of all PIPs that it can drive. As in the case of pruned wires, pruned PIPs are eliminating from tracking coverage.

All real wires and PIPs in the device are initially flagged as unused. The lists of wires and PIPs are stored in bit sets for rapid access and maximum efficiency.

As the interconnect coverage code visits every XDL PIP in the set of test designs, for every wire and PIP used in the design, the unused flag is cleared. The wires and PIPs still marked unused after inspection of each test suite are reported to the user. The final percent coverage for wires and PIPs is 100 % minus the number of uncovered wires in the device and 100 % minus the number of uncovered PIPs in the device.

### 6.3 Bitstream Coverage

Bitstream coverage determines how many bits in the configuration bit space have been covered by tests. This metric is not itself a primary result of the tests, but it serves as a sanity check for the other metrics. By the end of the test suites, we expect that nearly all PIP bits will have been touched, and that some reasonable subset of the logic setting bits will also have been touched.

In a hypothetical case where 100 % of logic settings and 100 % of PIPs were covered, we would expect to find nearly 100 % of the bitstream bits used. To not find a very high coverage of bitstream bits would imply that a correspondingly large percentage of logic settings or PIPs or undocumented features were not covered.

Bitstream coverage tracking begins by counting the number of configuration frames in the bitstream. It then looks up the number of bits in each configuration frame, and flags each bit as initially unused. The flags reside within an internal bitmap of structure comparable to the bitstream.

As the bitstream coverage code visits every bitstream in the test suite, for every configuration bit used in a bitstream, the corresponding unused flag is cleared. The bitstream flags still marked unused after inspection of each test suite are reported to the user. The final percent coverage is 100 % minus the number of unused bits divided by the total number of configuration bits in the bitstream.

## **6.4 Operation**

The coverage verification utility can be run on any combination of XDL files, bitstream files, and directories. Every directory encountered is scanned for subdirectories, and any XDL or bitstream files along the way are processed. When all relevant files have been processed, the final coverage metrics are presented to the user.

## 7 | Coverage

The test generation code was originally tested on a mid-sized XC7Z020 Zynq device. While computing the interconnect coverage metrics for the XC7K160T, a problem was uncovered with the interconnect test generation. More specifically, the test generation forced the desired PIPs into the test nets, but relied on `par` to create paths to and from them. In a large number of cases, `par` simply retained those PIPs while creating other paths, such that many of the PIPs were never exercised but would have reported success in hardware.

We believe that the interconnect test approach is sound, but the test generation depends too strongly on `par` to complete the nets without being able to adequately control `par`. Our team is working on correcting this, but it will very likely require developing a Torc-based router and route replicator instead of relying on `par`.

Table 7.1: Coverage results for XC7K160T.

Category	Covered	Uncovered	Total	% Covered	Sites Covered	% Covered
Logic Values	11,824,550	2,041,543	13,866,093	85.28	25,351 of 29,679	85.41
Category	Covered	Uncovered	Total	% Covered	Tiles Covered	% Covered
Tile Wires	10,490,907	7,484,047	17,974,954	<58.63	43,435 of 49,590	87.59
Tile Pips	37,696,479	30,030,267	67,726,746	<55.66	36,257 of 49,590	73.11
Category	Covered	Uncovered	Total	% Covered	Frames Covered	% Covered
Frame Bits	24,941,365	28,050,635	52,992,000	47.07	10,670 of 16,560	64.43

Table 7.1 shows upper bounds on wire and PIP coverage (58.63% and 55.66%), because we know that `par` is currently bypassing an unknown number of PIPs. Even if these percentages were correct, they are still very low and consequently explain why the percentage of covered frame bits is so low. The frame bit coverage would automatically increase with greater PIP coverage.

The percentages of tiles covered for wires and PIPs (87.59% and 73.11%) indicate how many tiles are impacted by the interconnect tests. This is a reflection of the number of tiles that contain SLICEL and SLICEM interconnect, and the number of neighboring tiles that support parts of the routing. These numbers can only increase if we provide coverage for additional logic sites or if we drive PIPs from other tiles and use special foldback PIPs.

It should be noted that there are actually 47,229 logic sites in the XC7K160T device, but 17,550 of those are TIEOFF sites that drive HARD1, HARD0, or WEAK1 and have no configuration settings. If these TIEOFF sites were included, then the percentage of total sites covered would be 53.68%.

## 8 | Conclusion

### 8.1 Revisions

The generation of interconnect tests needs to be rewritten as a Torc-based utility. For a given PIP that approach will allow us to create a single route from a slice to the PIP and from that PIP back to the slice. In some cases the route will spill out of the INT tile. As long as no route contains the same wire name in more than one tile, then the route can be replicated across all of the INT tiles. This revision is the most pressing and critical need in the independent functional testing effort.

Another feature that has not yet been implemented is the configuration memory testing. This is expected to be done through configuration and readback with a range of varying patterns.

### 8.2 Future Work

In addition to the revisions that need to be made, there are many other things that could be done to improve test scope and coverage and to reduce testing time.

#### 8.2.1 Other Logic Sites

Only SLICEL and SLICEM logic sites are being tested at present, with a resulting coverage of 85.28 % of all logic settings in the device. This number should increase into the 90 %–95 % range with the addition of BRAM and DSP sites.

#### 8.2.2 Additional Interconnect

Once the interconnect problems are resolved, the coverage can be pushed still higher by developing tests for the dedicated clock network. Further coverage would depend upon an in-depth analysis of what PIPs were still not being covered and a viable approach to include them—this is feasible but has not yet been investigated.

#### 8.2.3 Fault Isolation and Fault Diagnostics

Fault isolation and fault diagnostics were not in scope for this effort, but both could be performed in the future if necessary.

#### 8.2.4 Testing Time Reduction (PIP Packing)

Testing time is currently bounded by the number of PIPs in an INT tile. By creating paths with multiple PIPs, and by pruning tests with PIPs that are already fully covered elsewhere, we can significantly decrease the number of test bitstreams and consequently the testing time.

#### 8.2.5 Testing Time Reduction (Test Order Optimization)

By determining the Hamming distance between bitstreams, we can reorder the tests to reduce the number of frames that must be reconfigured from one test to another. This allows us to use partial bitstreams, where each partial is based on the difference from one bitstream to the next, and to further reduce the test time. We believe this approach to be very promising.

#### 8.2.6 Timing Verification

It is possible to test device timing by sweeping the clock frequency until failure and comparing that frequency to the expected fabric speed. This could be useful both for binning purposes and for helping determine whether the device under test is a counterfeit.



### **8.2.7 Shorting Fault Model**

Testing a device for shorts would be exponentially more complex than testing for stuck-at faults. An appropriate shorting fault model for the device would need to be developed, and every wire or PIP under test would need each of its neighbors biased with the opposite polarity. This is made significantly more complex by the fact that we don't know which wires or PIPs are adjacent at the VLSI layout level, so we would need to vastly over-specify the problem.

### **8.2.8 I/O Pin Testing**

Our testing has specifically assumed that I/O pins were unavailable. If we relax that constraint, there are many aspects of IOBs, SERDES, and pad I/O standards that could be tested with the help of a chip tester or a specially designed board.

# **ITAG FPGA UNDOCUMENTED FUNCTIONAL DISCOVERY FINAL REPORT**

**Matthew French, Andrew Schmidt, and Aravind Dasu**

**Information Sciences Institute  
University of Southern California  
3811 N. Fairfax Drive  
Suite 200  
Arlington, VA 22031  
{mfrench,aschmidt,dasu}@isi.edu**

**Modified: September 21, 2015**



---

<b>1</b>	<b>Executive Summary</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>4</b>
<b>3</b>	<b>Overview of Evaluated Hard IP</b>	<b>6</b>
3.1	Undocumented Functionality Description	6
3.2	Xilinx DSP48E Hard IP Block	6
3.3	DSP48E's Cascade Circuitry	7
3.4	DSP48E's ALU and Operation Mode Circuitry	8
<b>4</b>	<b>Technical Approach</b>	<b>9</b>
4.1	Knowledge-based Partitioning	9
4.2	Behavioral Modeling	10
4.3	On-Chip Circuit Analysis	12
4.4	Isomorphic Sub-circuit Extraction	13
4.4.1	Example 1	15
4.4.2	Example 2	15
4.5	Output	15
<b>5</b>	<b>Experimental Results</b>	<b>20</b>
5.1	Cascade Circuit Results	20
5.2	ALU and Operation Mode Results	21
<b>6</b>	<b>Conclusion</b>	<b>23</b>
<b>7</b>	<b>References</b>	<b>24</b>
<b>8</b>	<b>Appendix</b>	<b>25</b>
A	Cascade Circuit Full Results	25
B	ALU Op Mode Full Results	25

# 1 | Executive Summary

In this work we explored a novel approach combining on-chip execution and formal methods to exhaustively discover, explore, and describe the functionality of undocumented features in the DSP48E hard IP unit on the Xilinx Virtex5 devices. Using a knowledge based discovery approach, we **identified 1,518 undocumented modes** for this piece of IP. On-chip circuit analysis then **identified the functionality of 1,136 of these modes** and also discovered additional undocumented modes accessible through the bitstream. These previously undocumented modes are described at the **mathematical function level** in the appendix herein. These functions include the output of **partial products**, output of **intermediate shift register values**, output internal constants used by the circuit to perform **Boolean logic operations**, and several other functionalities. To provide a circuit level description of the functionality and to address scalability, our approach also utilizes an Isomorphic Sub-circuit Extraction technique based on formal methods to find and remove common circuits between the version of the circuit model derived from the documentation and the version of the circuit model derived from the empirical on-chip testing. The Isomorphic Sub-circuit Extraction technique proved to **reduce the evaluation state space by a factor of  $2^5$  to  $2^{15}$**  depending on the input circuits. Overall, this study was extremely effective and further research into evaluating other IP types or processor types is warranted.

## 2 | Introduction

Modern integrated circuit devices have become enormously complex. In scale, there are now many devices that are over 1 billion transistors in size. Additionally, with so many transistors available, silicon devices are largely complex System-on-Chip devices and becoming very heterogeneous in terms of underlying circuitry and features. In many respects, commercial Field Programmable Gates Arrays are at the forefront of these trends. They have had devices over 1 billion transistors shipped since 2014, and the number of heterogeneous Hard IP blocks available to an end user has steadily increased with each generation. Today, there are over 15 types of FPGA Hard IP features exposed to the user, and several more which only the vendors are aware of.

These undisclosed features have become more prevalent in the sub 65nm fabrication era. As fabrication costs have escalated with each node and pressures for time to market have increased, industry increasingly has used the current generation device to do trial runs of next-generation architecture features. If the feature works, they enable support for it in the current generation. If not, they deprecate the access to these features, usually through compilers or CAD tools, learn from their mistakes, and attempt an improved design in the next generation device. A prominent example of this is Xilinx's System Monitor, or SYSMON, hard IP block which is supposed to provide limited analog to digital conversion and temperature sensing. The block was implemented in VLSI for the Virtex-4, it was not implemented correctly, was deprecated from being enabled in Xilinx's CAD tools, and then was re-designed and supported in Virtex-5. The PowerPC cache parity circuit is another feature which was attempted in both the Xilinx Virtex-4FX and 5FX, but did not work in either generation, and access was disabled in software, even though the circuitry exists in hardware. Also, vendors will also seek to reduce NRE costs by re-using masks for similar products but enabling different features either through software support or packaging. A recent discovery was that both Xilinx and Altera do not tape out a new mask for each package size and instead deactivate I/O that are present in the mask, but not connected to the package. The CAD software that deactivates these can easily be circumvented and unbonded I/O can be driven. In addition, there are many built in self tests (BISTS) and other yield diagnostics that are built into devices that are not explained to the end user.

Usually these *undocumented features* are the product of industry operating in a highly cost competitive market, and these features are not inserted with malicious intent by the corporation. However, this does not preclude the event in our global market place that a foreign adversary cannot put in a malicious feature, or that a well intentioned errata does not result in a security vulnerability. In fact, there is a well known example of an FPGA vendor leaving in a backdoor to its bitstream through the JTAG interface [1]. Additionally in the radiation hardened Xilinx Virtex-5 part, the embedded PowerPC was disabled in the latter stages of development, leading to many open questions as to how completely it was disabled and if it could be somehow activated.

The military in particular is a heavy user of FPGAs. The Deputy Assistant Secretary of Defense (Systems Engineering) recently presented that 72% of DoD ICs are non ASICs, and that these are largely FPGA devices. The F-35 is comprised of over 200 FPGA devices, consisting of 64 different FPGA types, as compared to 9 different ASIC types [2]. However, the DoD now represents only a small fraction, <10%, of the FPGA industry's market. This makes it difficult for the DoD to have much influence into the security features or development processes that the FPGA industry adapts.

Given this environment, it is imperative that DoD have independent mechanisms to test and verify FPGA functionality, independent of the vendors, which is non-destructive, and scalable to billion transistor levels. Our approach leverages several key insights:

- Unlike other COTS processors (Intel, ARM, etc), FPGAs have a rich set of circuit level documentation in user's guides and patents that can bootstrap knowledge about the underlying circuitry to a great extent.
- FPGA devices are fully programmable, meaning using custom tools such as USC/ISI's Torc tools, the physical device can be extensively probed and intentionally set into undocumented modes to determine undocumented outputs.

- Formal methods typically utilized for circuit validation can be adapted to finding differences between documented and observed behaviors.

These insights are especially true if the end goal is to identify undocumented functionality, and not the impossible problem of finding differences in exact implemented circuitry when the true implemented circuit is not known. USC/ISI's approach, detailed in Figure 2.1, consists of four key stages: Knowledge-based Partitioning, Behavioral Modeling, On-chip Circuit Analysis, and Isomorphic Sub-circuit Extraction.

In Knowledge-based Partitioning, a thorough analysis of the known sources is performed for the FPGA's hard IP, such as user manuals and patents. From this analysis a Behavioral Model can be developed and refined to describe the vendor specified functionality of the hard IP. The analysis also provides insight into how to perform On-chip Circuit Analysis to develop an empirical circuit model that more accurately reflects the functionality of the actual hard IP circuit when running in states not allowed, or supported, by the vendor. Comparing the resulting two models can yield an extremely large state space search. While Knowledge Based Partitioning initially subdivided this problem into valid and unspecified functional modes, a further state space reduction is required. To address this the last stage in our approach uses Isomorphic Sub-circuit Extraction with graph-based formal methods, to determine equivalent circuits and remove them from the search space. The result is the final difference in the two circuits, or in this case the undocumented functionality.

For this work, the DSP48Es in the Xilinx Virtex-5 devices were selected as the hard IP under investigation. The following sections provide an overview of the DSP48E's hard IP module (Chapter 3), a more detailed description of the technical approach (Chapter 4), the experimental results (Chapter 5), and a summary of our findings (Chapter 6).

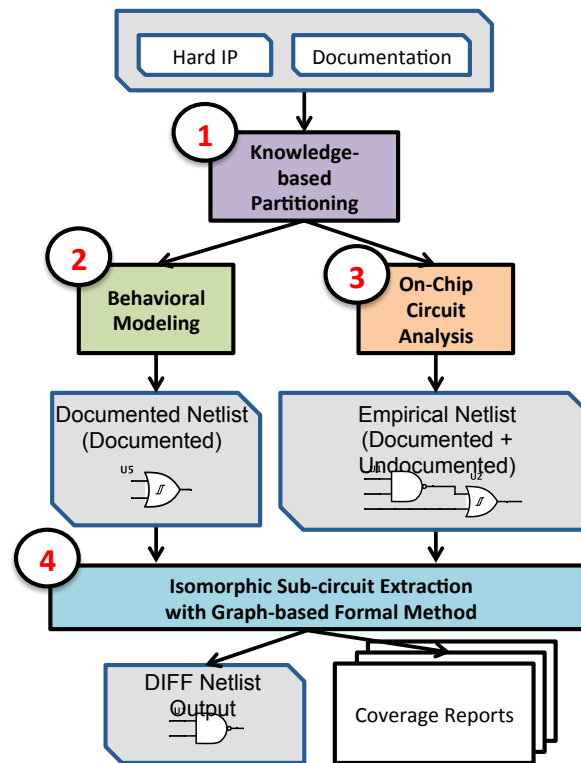


Figure 2.1: USC/ISI's Functional Discovery Tool Suite

## 3 | Overview of Evaluated Hard IP

### 3.1 Undocumented Functionality Description

Under this study, the DSP48 hard IP in the Xilinx Virtex-5 devices is used as a test case for discovering undocumented functionality. Undocumented functionality refers to the behavior of the evaluated IP when operating in modes that are explicitly listed as illegal or invalid by the vendor's user guides and documentation. Moreover, in certain cases the vendor may omit the behavior of the IP by not documenting that additional modes even exist. In all of these cases this study considers when both the input modes and the output behavior are not defined by the vendor, the resulting functionality is undocumented.

### 3.2 Xilinx DSP48E Hard IP Block

The DSP48E is a hard IP block implemented in the VLSI of the FPGA device of the Virtex-5 FPGA, seen in Figure 3.1. The DSP48E block was selected for investigation because it has a long heritage across FPGA families, and is of a moderate sized complexity for a reasonably sized study. Multiplier units were first introduced in the Virtex-2 series. Each new generation of Virtex devices has seen the multiplier unit become more complex and adding new functionality, to the point where in Virtex-5 they were renamed to DSP blocks. The Virtex-5 DSP block includes multiplication, multiply and accumulate (MACC), three-input add, barrel shift, wide-bus multiplexing, magnitude comparator, bit-wise logic functions, pattern detection, and wide counters.

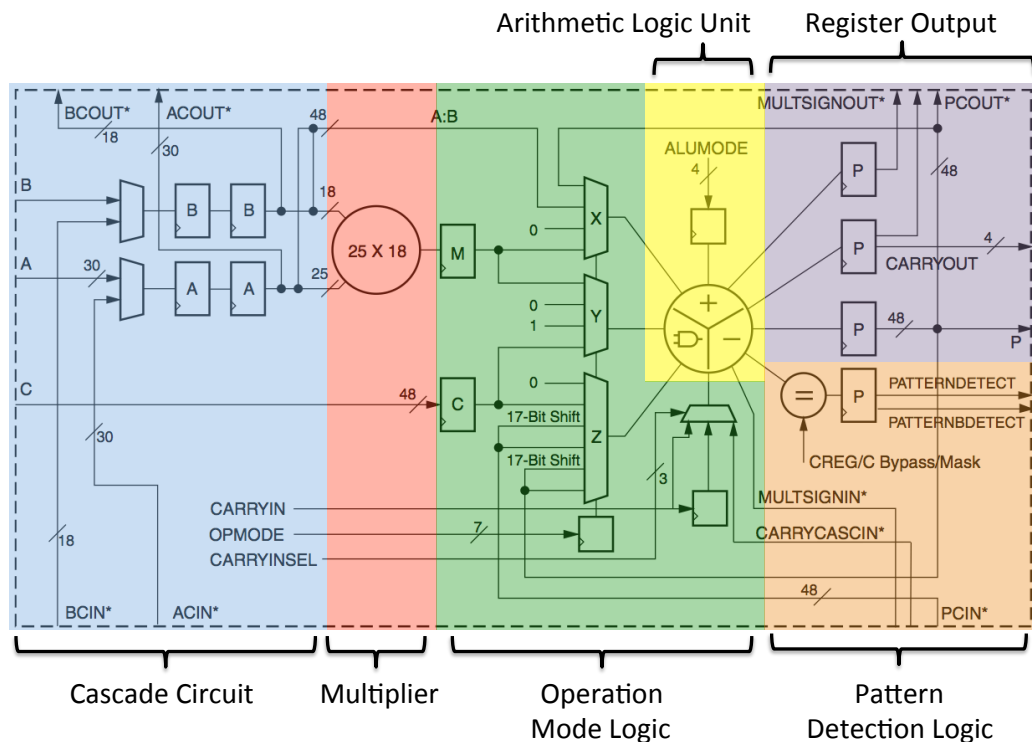


Figure 3.1: Block diagram of the Virtex-5 DSP48E IP broken into sub-circuits

The architecture also supports cascading multiple DSP48E slices to form wide math functions, DSP filters, and complex arithmetic without the use of general FPGA fabric [3]. With this heritage, the DSP is a good candidate to contain either new features that the vendor decided to be pushed to Virtex-6, or legacy features and errata from

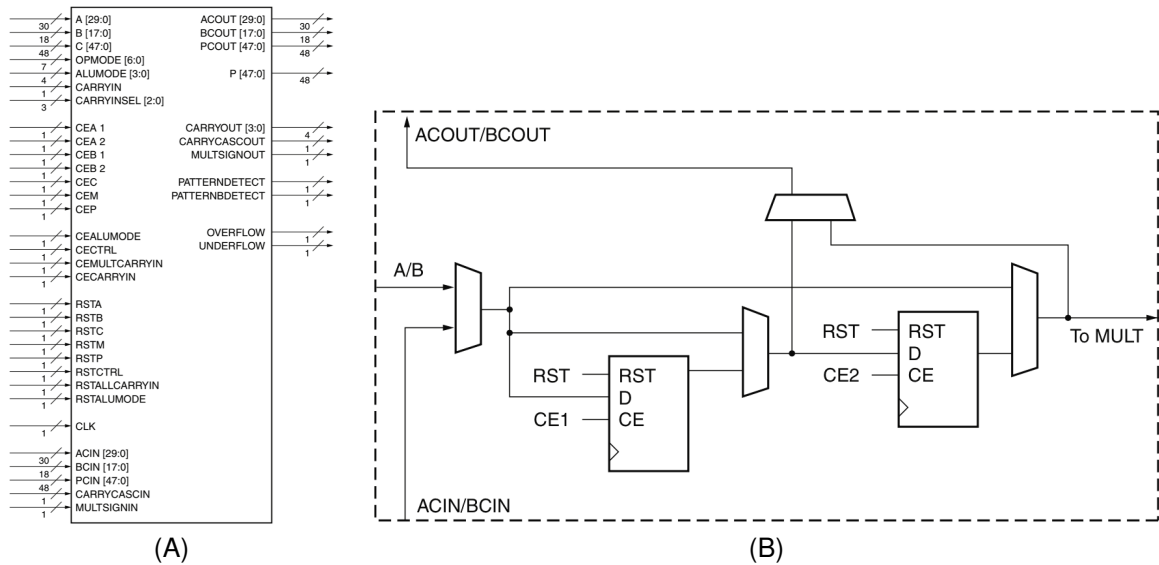


Figure 3.2: (A) Primitive diagram for DSP48E top-level I/O and (B) cascade sub-circuit circuit block diagram

previous generations that due to design costs were not cleanly re-designed. The size of this IP block is also ideal for this initial study as it is much more complex than the programmable fabric, but less complex than other hard IP on the device, such as the EMACs or embedded processors. Finally, as described below, even a cursory glance at the DSP48 user's guide reveals several undocumented modes.

In the Virtex-5 XC5VLX110T FPGA that is on the XUPv5 Development Board[4] there are 64 DSP48E blocks spanning a single column in the device. In the largest Virtex-5, the XC5VSX240T FPGA, there are 1056 DSP48E blocks split across 11 columns. Techniques developed as part of this work have been setup to support any Virtex-5 device, and have are scalable to evaluate multiple DSP blocks in parallel, at run-time. The DSP48E primitive, implemented in a user design, has 335 input/output signals which include both control and data signals. A majority of these signals include the A, B, and C input operands and P output resultant, shown in Figure 3.2(A).

### 3.3 DSP48E's Cascade Circuitry

The DSP48E block's functionality for certain behaviors, such as cascading pipeline registers, is explicitly configured through the primitive parameter settings. During the design's implementation, these parameters are turned into bitstream configuration bits. Unlike data or control inputs, the configuration bits do not change during the design's run-time. An example of the parameters used in the cascade section of the DSP is shown in Table 3.2(B), which comes from the Xilinx User Guide 193, Table 1-5.

From Table 3.1 it is observed that the User Guide only describes valid states for *AREG/BREG* and *ACASCREG/B-CASCREG*, along with the expected functionality. There is no mention of the functionality when using undocumented parameters. An undocumented example would be assigning 0 to *AREG* and 1 to *ACASCREG*. In fact, of the nine possible settings, only four are listed in the User Guide as being valid, the remaining five are not specified or not allowed by the conventional vendor tool flow. The true behavior of the functionality can be hypothesized by looking at the detailed circuit, illustrated in Figure 3.2(B); however, this level of documentation is not always provided, and it may be incomplete. Table 3.2 lists the legal modes for a given configuration combination for the A and B cascade circuits of the DSP48E block. Overall, the ten undocumented modes for A and B input's as part of the cascade circuit will be evaluated as part of this effort, five modes for A and five modes for B. The results of the study of the undocumented modes of the cascade register are presented in Chapter 5.



Table 3.1: DPS48E Cascade circuit valid configuration settings from Xilinx UG193

Pipeline Registers		Notes (Refer to Figure 1-7)
AREG, BREG	ACASCREG, BCASCREG	
Current DSP	To Cascade DSP	
0	0	Direct and cascade paths have no registers.
1	1	Direct and cascade paths have one register.
2	1, 2	When direct path has two registers, cascade path can have one or two registers.

**Note:** If AREG = 1, then CEA2 is the only clock enable pin that is allowed to be used. If AREG = 0, then neither CEA1 nor CEA2 should be used. If AREG=2, then CEA1 and CEA2 can be used where CEA2 is the clock enable for the second register. This holds true for BREG and CEB1/CEB2 enable pins.

Table 3.2: A and B cascade register parameters based on UG193

REG	CASCREG	Mode
0	0	Legal
0	1	Undocumented
0	2	Undocumented
1	0	Undocumented
1	1	Legal
2	0	Undocumented
2	0	Undocumented
2	1	Legal
2	2	Legal

Table 3.3: DPS48E Operating Mode control bit select Z multiplexer configuration settings from Xilinx UG193

Z OPMODE[6:4]	Y (OPMODE[3:2])	X (OPMODE[1:0])	Z Multiplexer Output	Notes
000	xx	xx	0	Default
001	xx	xx	PCIN	
010	xx	xx	P	
011	xx	xx	C	
100	10	00	P	Use for MACC extend only
101	xx	xx	17-bit Shift(PCIN)	
110	xx	xx	17-bit Shift(P)	
111	xx	xx	xx	Illegal selection

### 3.4 DSP48E's ALU and Operation Mode Circuitry

This work observes that in addition to several cascade circuit settings the run-time values for the ALU and Operation (Op) mode inputs are not fully specified by the vendor. From Figure 3.2(A) the total number of bits for the ALU mode is 4 and the Op mode has 7. This leaves 2048 ( $2^{11}$ ) possible combinations for the ALU and Op mode that would need to be specified in order to have no undocumented modes. Unlike the cascade circuit, these run-time parameters are user specific and can change during the execution of the design at run-time. From UG193 only a subset of the 2048 combinations are actually specified. In fact, only 1508 valid modes are documented. Table 3.3 further illustrates this by explicitly showing the *Illegal selection* note (from Table 1-8 in UG193) for OPMODE settings with respect to the output for the Z multiplexer. Especially for control signals, such as the OPMODE, the vendor tools have no ability to check designs at run-time to verify proper usage. Instead, if the OPMODE[6:4] bits are set to '111' the user has no guarantee what the output of the Z multiplexer will be. In Chapter 4 the technical approach for not only identifying the number of undocumented modes, but also their behavior is presented.

## 4 | Technical Approach

As previously mentioned, USC/ISI's approach, detailed more fully in Figure 4.1, combines four key stages: Knowledge based Partitioning, Behavioral Modeling, On-chip Circuit Analysis, and Isomorphic Sub-circuit Extraction, in order to identify undocumented functionality and extract differences in implemented circuitry. Knowledge-based Partitioning, Behavioral Modeling, and On-chip Circuit Analysis are utilized to develop models of the FPGA hard IP from known sources and then compared against the actual behavior of the device while in operation. Isomorphic Sub-circuit Extraction uses graph-based formal methods to determine equivalent circuits and remove them from the search space, with the result being the final difference in the two circuits. In effect, these differences are the undocumented functionality. The following sections provide more detail on each processing step.

### 4.1 Knowledge-based Partitioning

The analysis begins with knowledge based partitioning. The IP block is manually subdivided into units that can be further reduced in complexity. Figure 4.2 illustrates the basic flow that is taken for the DSP48E hard IP block. This is accomplished through the analysis of documentation, user guides, patents, or any vendor provided simulation models which suggest sub-block functionality of the IP. Fortunately, FPGA documentation is largely provided at the circuit level, so it is easily decomposable into viable sub-blocks, from which documented behavioral models are constructed. The DSP48E blocks have been presented in Figure 3.1. The approach is well suited for FPGAs which are developed with modularity in tile type (Slice, BlockRAM, DSP etc...). For the DSP48E, the raw functionality is broken down into the following atomic units: cascade circuitry, 25-bit×18-bit multiplier, operation mode logic, arithmetic logic unit, register output, and pattern detection logic, as highlighted in the figure. Behavioral models replicating the documented behavior are then developed manually.

The Knowledge-based partitioning stage also yields a summary of the number of undocumented modes there are for the DSP48E, shown in Table 4.1. The rest of this work will describe the efforts to describe the behavior of these modes.

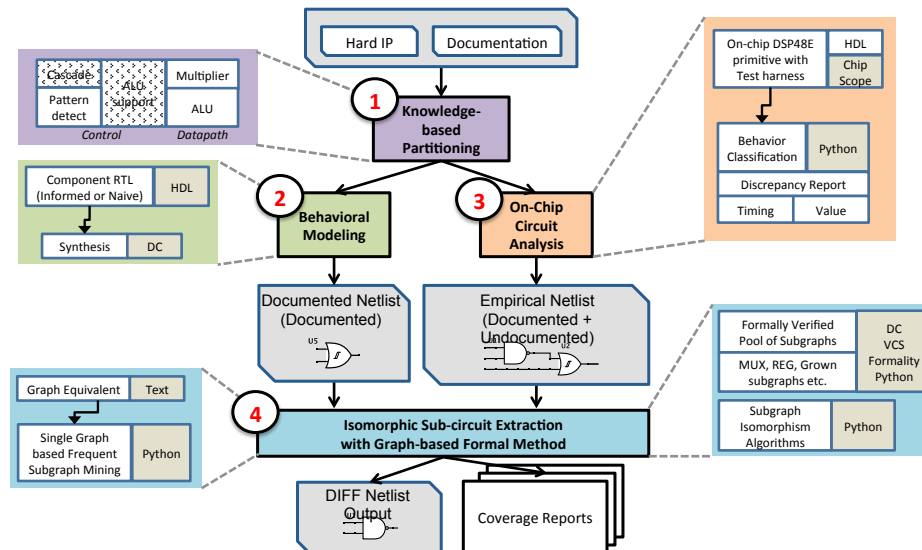


Figure 4.1: ISI's functional discovery tool

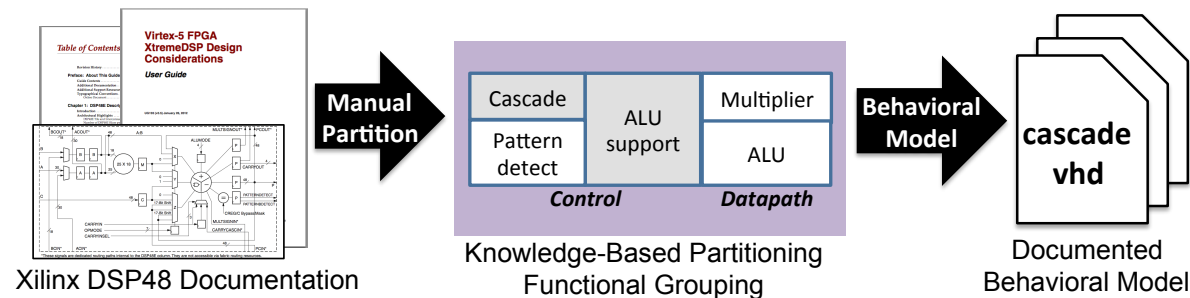


Figure 4.2: Knowledge-based Partitioning Flow

Table 4.1: Summary of undocumented modes from knowledge based partitioning

DSP48E sub-circuit	Undocumented Modes
<b>Cascade Register:</b>	
A Input sub-circuit	5
B Input sub-circuit	5
<b>Operation Mode Logic:</b>	
X Operand Multiplexer	268
Y Operand Multiplexer	512
Z Operand Multiplexer	728
<b>Arithmetic Logic Unit:</b>	* Evaluated with of Op Mode *
Register Output	0
Pattern Detection Logic	0
Multiplier	0
<b>Total Undocumented Modes:</b>	<b>1518</b>

4.2 Behavioral Modeling

Once the atomic sub-blocks have been identified, behavioral models are created. Presently, this is a manually process which involves constructing VHDL and Verilog simulation models, independent of any vendor models. Since the behavioral models rely on documented information describing the functionality, the model is intended only to capture valid modes specified by the vendor. An example of the DSP48’s cascade circuit model is shown in Listing 4.1. Test vectors are used to validate the model using commodity tools to perform automated test pattern generation. The model is synthesized using Synopsys Design Compiler to generate a netlist to be used by the isomorphic sub-circuit extraction stage.

Listing 4.1: Example Verilog code produced as part of the Knowledge-based Partitioning Flow

```

1 // =====
2 // Behavioral model of DSP's cascade circuit
3 // =====
4 module cascade_dsp
5 (
6     clock,          // clock
7     reset,          // reset
8     enable,         // enable
9     in,             // data input
10    reg_ctrl,        // REG config
11    creg_ctrl,       // CASCREG config
12    out,             // Output to MULT
13    cout            // COUT
14 );
15
16 // Input ports
17 input clock, reset, enable;
18 input [2:0] in;
19 input [1:0] reg_ctrl, creg_ctrl;
20
21 // Output ports
22 output [2:0] out, cout;
23
24 // Wires and regs
25 wire clock, reset, enable;
26 wire [2:0] in, out, cout;
27 wire [1:0] reg_ctrl, creg_ctrl;
28 reg [2:0] r0, r1;
29 wire [2:0] g0, g1, g2, g3;
30 wire [2:0] rg0, rg1;
31 wire sel0, sel1, sel2;
32 reg sel0_r, sel1_r, sel2_r;
33
34 // Input output
35 assign g0 = in;
36 assign out = g2;
37 assign cout = g3;
38
39 // Register outputs
40 assign rg0 = r0;
41 assign rg1 = r1;
42
43 // Mux selects
44 assign sel0 = sel0_r;
45 assign sel1 = sel1_r;
46 assign sel2 = sel2_r;
47
48 // Muxes
49 assign g1 = sel0 ? rg0 : g0;
50 assign g2 = sel1 ? rg1 : g0;
51 assign g3 = sel2 ? g2 : g1;
52
53 // Control logic for mux selects
54 always @ (*)
55 begin
56     if (reg_ctrl == 2'b00 && creg_ctrl == 2'b00) begin
57         sel0_r <= 0;
58         sel1_r <= 0;
59         sel2_r <= 0;
60     end
61     else if (reg_ctrl == 2'b01 && creg_ctrl == 2'b01) begin
62         sel0_r <= 0;
63         sel1_r <= 1;
64         sel2_r <= 1;
65     end
66     else if (reg_ctrl == 2'b10 && creg_ctrl == 2'b01) begin
67         sel0_r <= 1;

```

```

68         sel1_r <= 1;
69         sel2_r <= 0;
70     end
71     else if (reg_ctrl == 2'b10 && creg_ctrl == 2'b10) begin
72         sel0_r <= 1;
73         sel1_r <= 1;
74         sel2_r <= 1;
75     end
76 end
77
78 // Registers
79 always @ (posedge clock)
80 begin
81     if (reset == 1) begin
82         r0 <= 0;
83         r1 <= 0;
84     end
85     else if (enable == 1) begin
86         r0 <= g0;
87         r1 <= g1;
88     end
89 end
90 endmodule

```

### 4.3 On-Chip Circuit Analysis

In order to understand the actual functionality of the IP block, USC/ISI utilizes on-chip circuit analysis to selectively configure, probe, and analyze the empirical behavior. A suite of tools has been developed to identify illegal bitstream configurations for the IP block and to provide run-time testing on an actual device. The tool flow is presented in Figure 4.3. In a form of reverse validation, the bitstream configuration settings for the IP are isolated to determine if any additional parameters are possible beyond what is suggested in the IP's user guide. An example of this would be if the user guide covered 7 configurations, which requires 3-bits ( $2^3=8$  settings), leaving one setting unaccounted for. The tool flow identifies the missing setting and generate a bitstream for on-chip testing to compare against the behavior model. While this is a general example, Chapter 5 presents results of this tool to uncover undocumented behavior with the DSP48 Hard IP block.

ISI's developed on-chip circuit analysis tool flow consists of the normal Xilinx Development Flow, which includes *Synthesis*, *Implementation*, and *BitGen*. This flow can be implemented in a conventional ISE project or through the commandline via a Makefile. The output of the normal flow is the *initial.bit* bitstream that will perform the original design behavior. The design can be modified after the *Implementation* stage leveraging Torc to change the Hard IP block's configuration attributes. Once the modifications are made *BitGen* is again performed to create new *modified.bit* bitstreams. The Bitstream Diff tool is then used to compare the different bitstreams to identify which bits are not covered by the available configurations of the IP block. These locations are stored for further

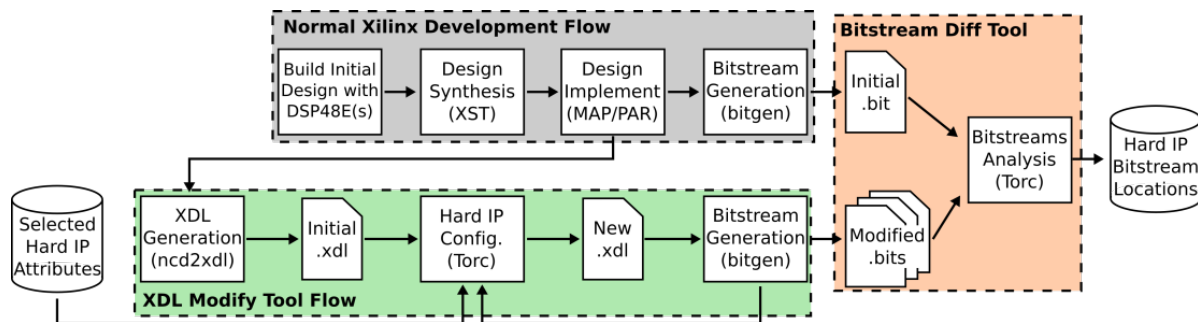


Figure 4.3: On-chip circuit analysis tool flow

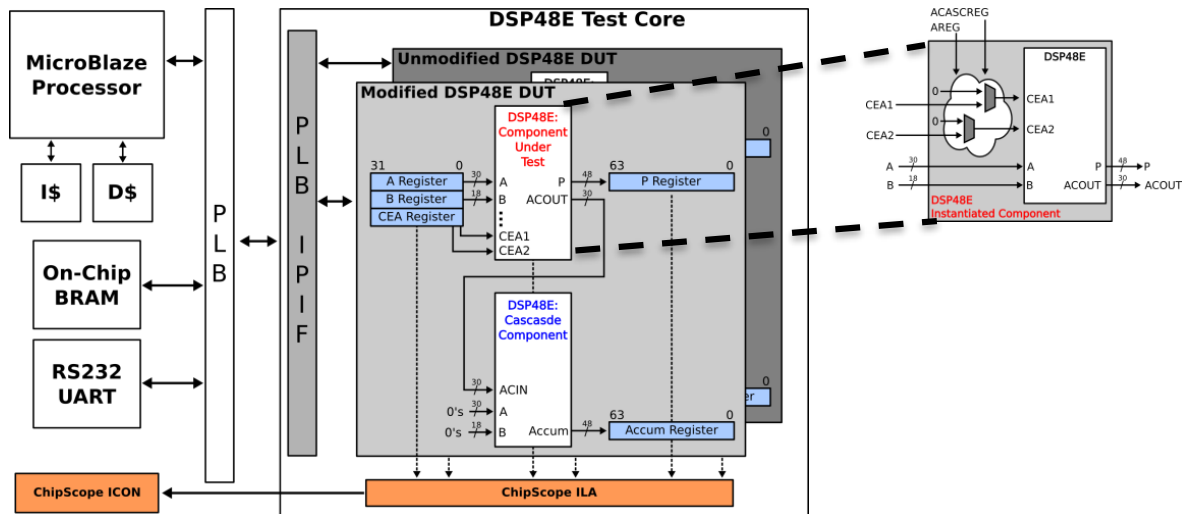


Figure 4.4: On-chip circuit analysis run-time infrastructure for DSP48E evaluation

analysis during the on-chip run-time testing, described next.

In addition to illegal bitstream configurations, the knowledge based partitioning provides undocumented modes from the IP block's datasheets, user guides, and patents. A majority of these modes are configurable at run-time, requiring a sophisticated on-chip testing infrastructure. ISI has developed an extensive testing methodology to evaluate the DSP48 block for the purposes of this work. This infrastructure is depicted in Figure 4.4. The on-chip testing leverages active partial reconfiguration to selectively re-configure just the bitstream configuration corresponding to the IP block under test to accelerate the overall testing. During each test run-time data is collected to provide insight into the outputs of the experiment. In the example of the DSP48 block the outputs of the *product* and *accum* register are stored by the MicroBlaze processor in memory. Upon the test's completion this run-time data is collected and analyzed through functionality scripts to determine whether the probed behavior matches the expected hypothesized behavior or is unexpected behavior. The *behavioral model* developed during the Knowledge-based partitioning is then updated to reflect the changes based on the on-chip testing to generate the *empirical model*. These two models are then used in the Isomorphic Sub-circuit Extraction stage.

#### 4.4 Isomorphic Sub-circuit Extraction

The state space reduction approach relies on the hypothesis that given a representative netlist (empirical), if the known fundamental-circuits in the netlist can be identified and formally verified, then significant state space reduction can be achieved. This would then enable further/future reverse-validation techniques to inspect the remaining netlist components for malicious/undesired behavior. Towards this, a graph mining algorithm and tools have been developed to search for instances of known fundamental-circuit structures (such as multiplexers) in a larger netlist (such as the cascade circuit, or Operation model logic in a DSP module), in order to achieve state space reduction and formal verification. The single graph based frequent sub-graph mining (SGFSM) algorithm, is shown in Figure 4.5.

The algorithm consumes two netlists: (a) The fundamental module/circuit to be mined, for example a 2:1 Mux in the form of a synthesized Verilog netlist, and (b) a large netlist that is expected to contain one or more instances of the fundamental module. For example, this can be a component of the DSP48E of the Virtex-5 FPGA, such as the cascade component, which contains three instances of a 2:1 Mux along with several registers and other control circuits (as previously seen in Figure 3.2(B)).

The synthesized netlist of the fundamental module (termed as the 'small' netlist/graph) is initially seeded, by selecting the net with the largest connectivity. Often this is the net with the largest fan-out. This seed, initial

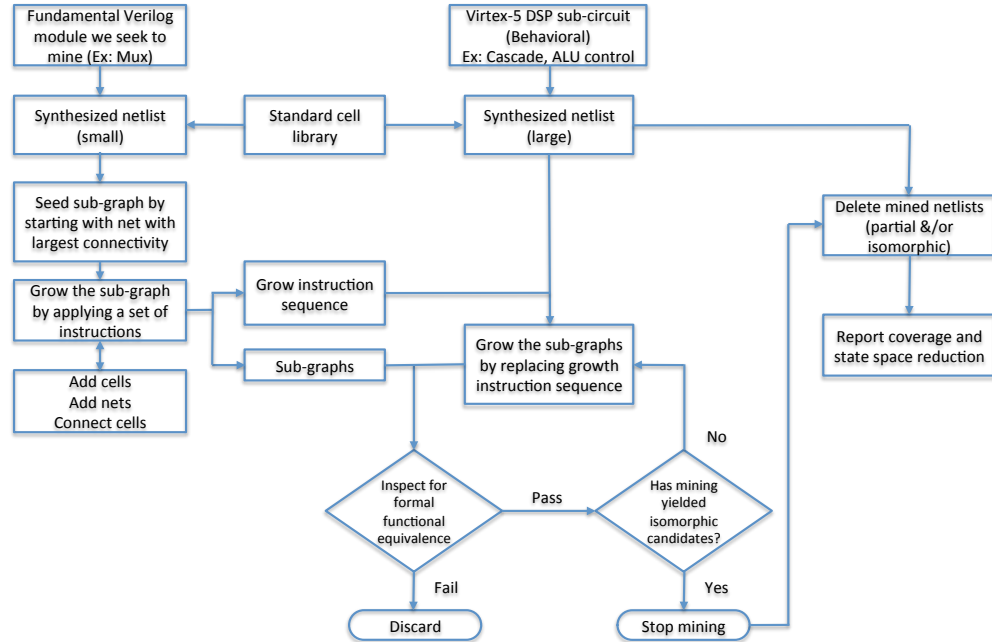


Figure 4.5: Flow chart of the single graph based frequent sub-graph mining algorithm

sub-graph, is then grown into larger sub-graphs, by applying a set of instructions: Add cells, Add nets, and Connect nets. Through this process, the sequence of instructions and the resulting suite of sub-graphs are recorded/memorized for processing the larger netlist.

The next step in the algorithm seeks to find potentially identical copies of the seed net in the large netlist/graph. This initial search does not seek anything other than a wire with the same number of connections, regardless of the gates/std-cells that it connects to. Next, the sequence of growth instructions previously memorized, is applied in a formal verification loop. I.E. each time the tuple of instructions (add cells, add nets, and connect cells) is applied on the potentially-identical- seeds of the larger netlist, the sub-graph (as a Verilog netlist) is compared against the peer sub-graph from the small netlist.

The comparison is performed using Synopsys formal verification tool, Formality. A caveat to note is since Formality requires explicit binding of either input ports, or output ports prior to a formal verification process, the algorithm involves an implicit port binding process. A second caveat to note is that Formality, primarily intended for minor circuit changes (known as engineering change orders), is being used for entirely different purposes. This poses a challenge of mimicking routine Formality user practices, such as absorbing inverters at outputs or inputs to mitigate logic inversions. The algorithm uses an implicit process (a Python script) to automatically explore or discard the process of inverter absorption.

If the formal verification process passes the candidate sub-graph, then it is inspected for isomorphism. This implies that a perfect and complete match has been found in the large netlist. If this check yields an incomplete match, the algorithm continues to iterate through the sub-graph growth process, until either a match is found or the growth stalls due to a complete failure to grow any further. At this point, the mining process stops and the matched sub-graphs (partial or isomorphic) are deleted from the large netlist. This process reduces the state space of the large netlist, thus allowing for either a small state space based manual/alternate inspection for functional mismatch, or subsequent mining of other fundamental modules. The algorithm terminates by generating a report of the candidates mined, and coverage obtained.

To better illustrate how the Isomorphic Sub-circuit Extraction works. We provide two examples of the process operating on the ALU-input control circuit and the cascade control circuit of the DSP48E.



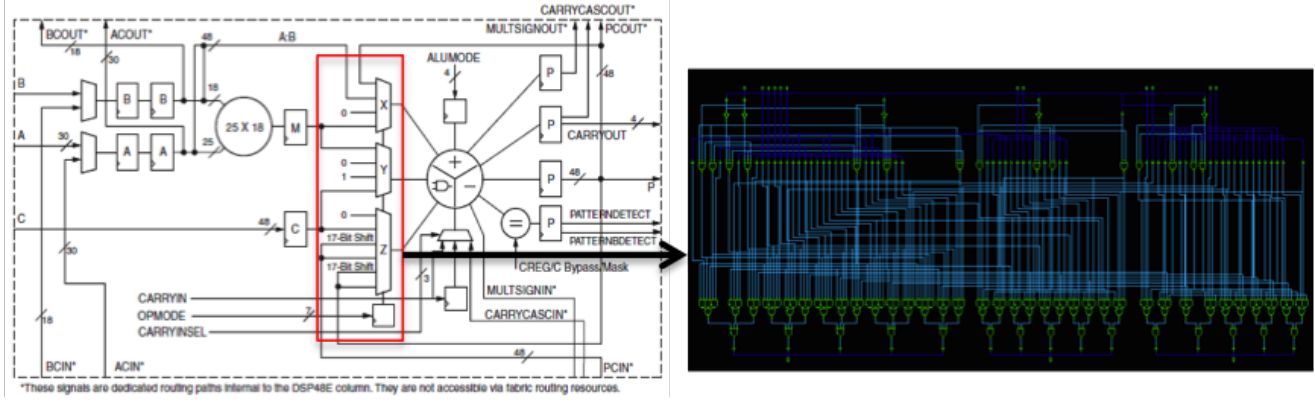


Figure 4.6: ALU-input control circuit of DSP hardIP in Virtex-5 FPGA and its synthesized netlist (with Mux resolution limited to 5-bit, for brevity)

#### 4.4.1 Example 1

In this example, the large netlist under consideration is the ALU-input control circuit (highlighted in red in Figure 4.6) and its synthesized version is obtained through Synopsys Design Compiler using a 45nm standard cell library is also shown in Figure 4.6. It should be noted that the Virtex V is 90nm technology. We utilized 45nm as that was the cell library available to us under this effort, and as our techniques are focused on identifying behavior, we are interested in a representative cell library, not the exact cell library. The behavioral model used to generate the large netlist, leveraged the discovery of the undocumented features via the on-chip testing methodology and knowledge-enhancement from relevant Xilinx patents. Specifically, the discrete distributions of the control signals to the three multiplexers (X, Y, Z) and the redundant case of 110 and 111 for the select lines of the Z-Mux were considered. The small netlist used to reduce the complexity of the larger circuit was then a 4:1 Mux (shown in Figure 4.7).

Figure 4.8(A) then shows how the 4:1 Mux was seeded in the ALU-input control circuit in the first growth sequence and Figure 4.8(B) shows the resulting fully grown sub-graph. The result of the mining (deletion of isomorphic circuits) is shown in Figure 4.9. The resulting netlist belongs to the 5:1 Z Mux, achieving a state space reduction of  $[2^{15}]$ .

#### 4.4.2 Example 2

The second example uses the cascade control circuit (highlighted in red in Figure 4.10) as the large circuit in the Isomorphic Sub-circuit Extraction process. The synthesized version obtained through Synopsys Design Compiler using a 45nm standard cell library is also shown in Figure 4.10. The behavioral model used to generate the large netlist, leveraged the discovery of the undocumented features via the on-chip testing methodology and knowledge-enhancement from relevant Xilinx patents. The small netlist used to reduce the complexity of the larger circuit was then a 2:1 Mux (shown in Figure 4.11).

The result of mining (isomorphic candidates are deleted) the 2:1 mux from the cascade control circuit is shown in Figure 4.12. The resulting netlists belong to the registers and other control circuits. Therefore for each 2:1 5-bit Mux that was mined, the algorithm achieved a state space reduction of  $[2^5]$ .

### 4.5 Output

Overall, the Isomorphic Sub-circuit Extraction can be used in this manner to remove common elements from the documented netlist resulting from the Behavioral Modeling stage and the empirical netlist resulting from the On-chip Circuit Analysis stage. The result, is a behavioral description of the undocumented functionality of the circuit.



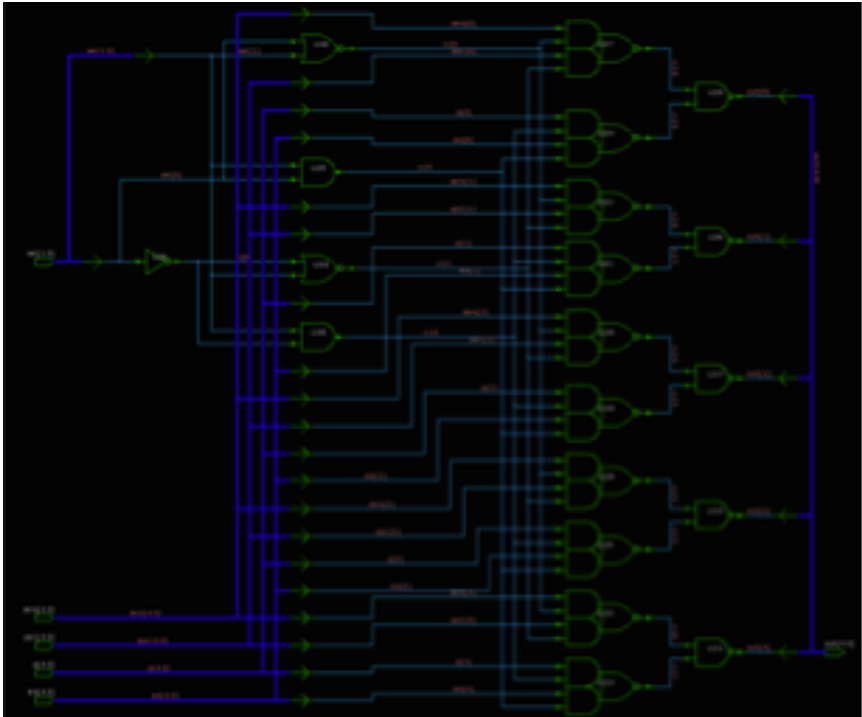
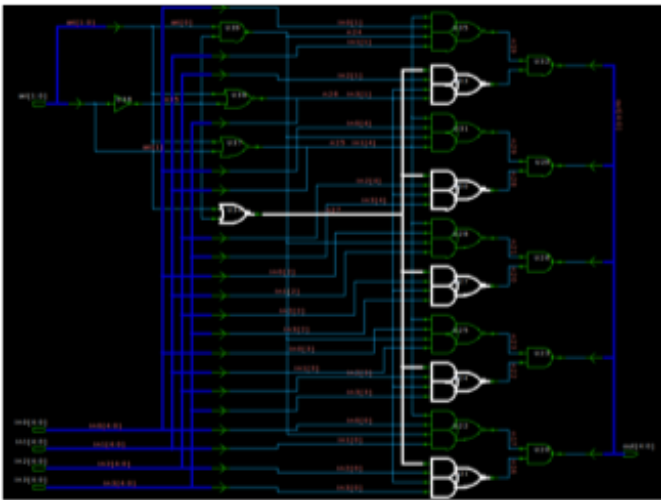
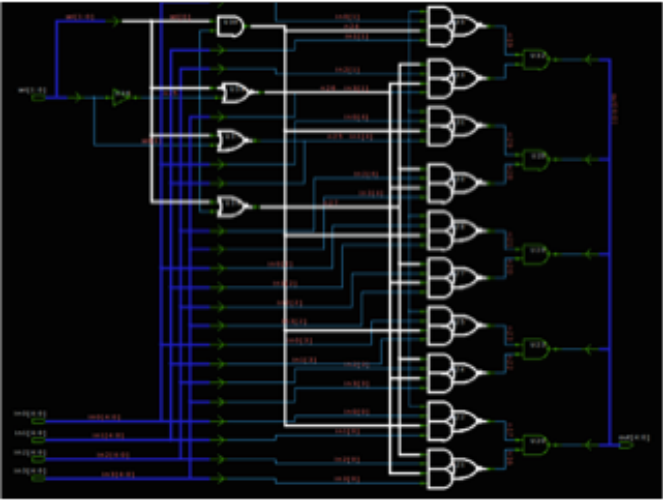


Figure 4.7: Synthesized small netlist of a 4:1 Mux (5-bit)



(a)



(b)

Figure 4.8: (a) Seed net and cells from initial sub-graph growth sequence and (b) Sub-graph after two iterations of growth sequence

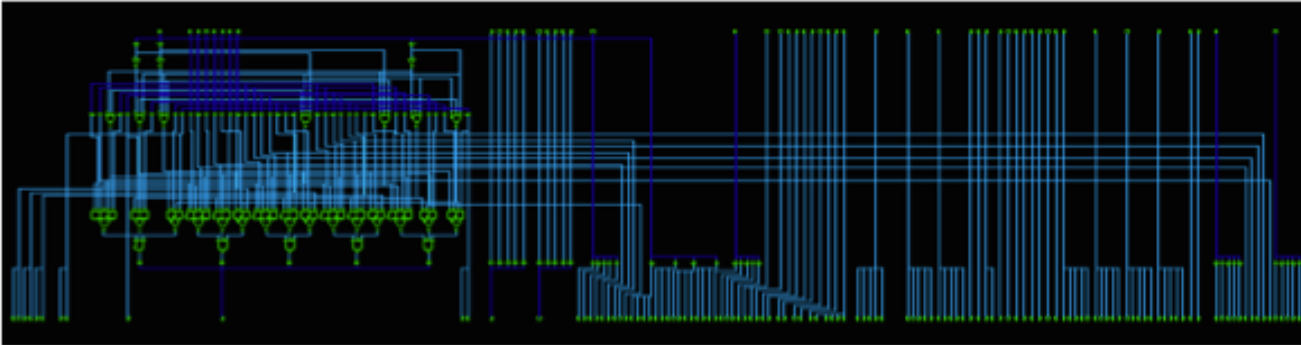


Figure 4.9: Isomorphic 4:1 Multiplexers mined and deleted from the ALU-input control circuits netlist

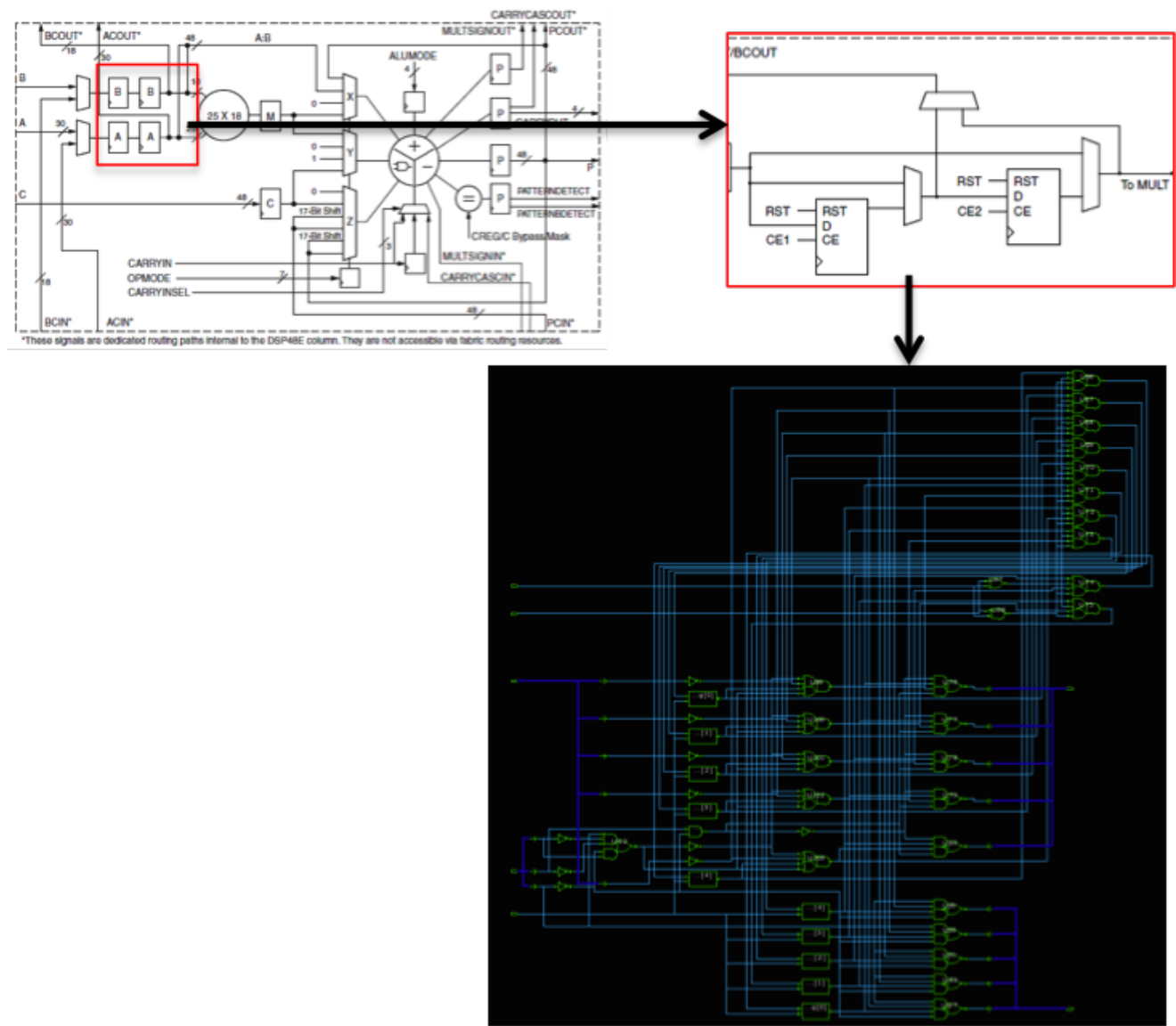


Figure 4.10: Cascade control circuit of DSP hardIP in Virtex- 5 FPGA and its synthesis

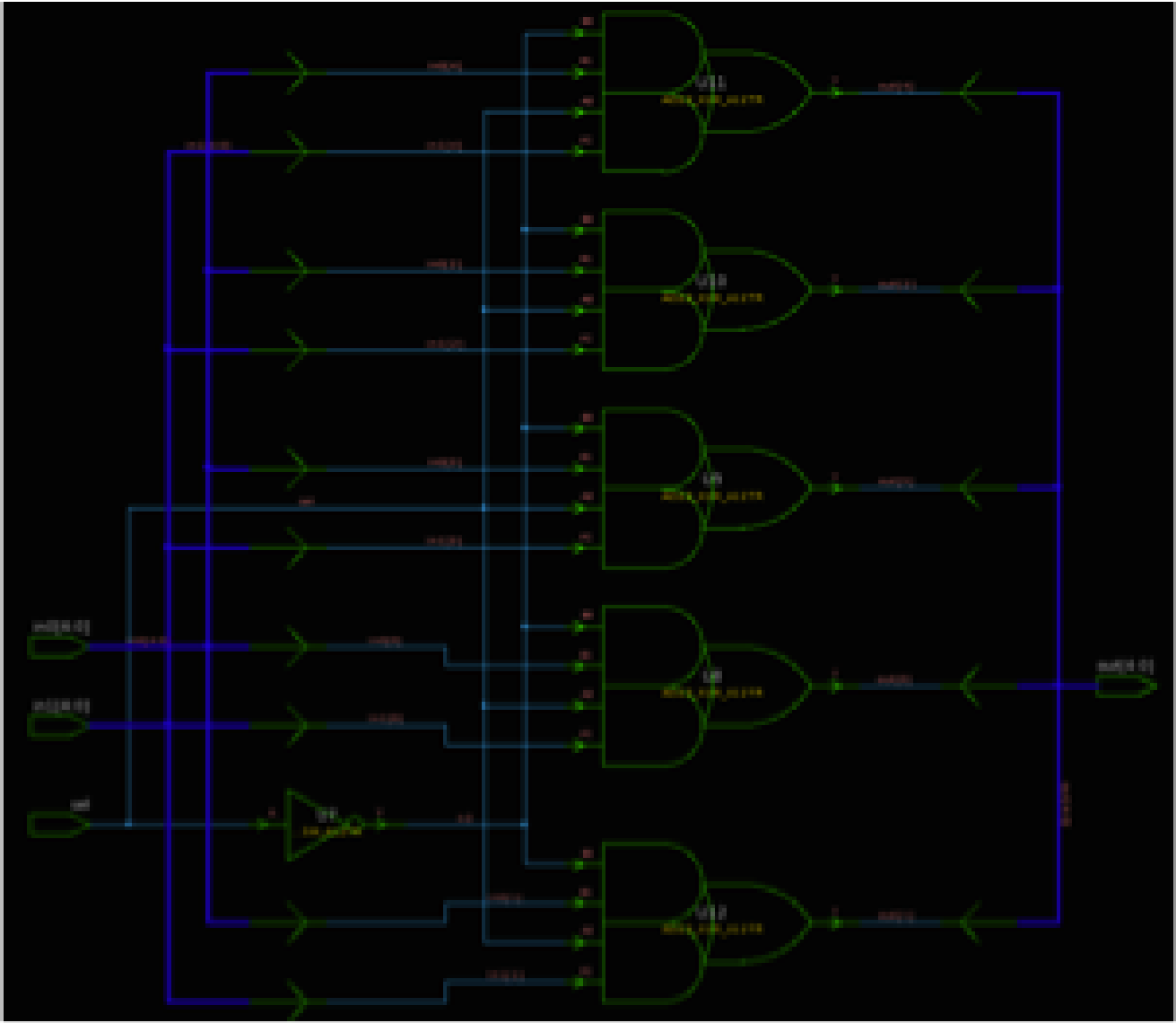


Figure 4.11: Synthesized small netlist of a 2:1 Mux (5-bit)

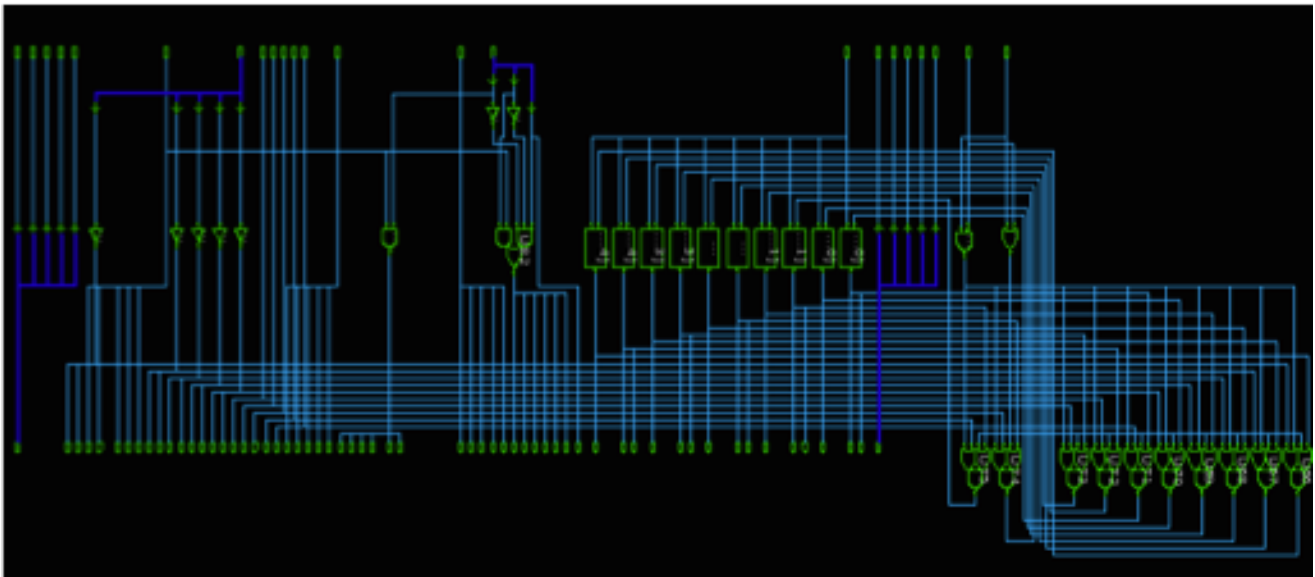


Figure 4.12: Isomorphic 2:1 Multiplexers mined and deleted from the Cascade control circuits netlist

## 5 | Experimental Results

Using the developed on-chip testing infrastructure a total of 1136 out of the 1518 undocumented modes have been tested on the DSP48E block of a Virtex5 FPGA. This covers 74.8% of the undocumented states, as seen in Table 5.1. The remaining 25.2%, predominately on the Z Operand Multiplexer, have been intentionally ignored as the scope of this effort was only to evaluate 33.3% of the undocumented modes and was not due to any underlying technical approach issue. The techniques developed as part of this effort could be used to finish evaluating the unevaluated modes as part of future work.

This study uncovered several interesting results. To briefly summarize them, the cascade circuit was originally considered to only consist of five undocumented modes, as described in Section 3.3. However, through bitstream analysis two additional undocumented modes were discovered, resulting in seven evaluated undocumented modes for each of the A and B cascade circuits, more details of which are presented in Section 5.1.

In addition to the cascade register, the ALU and Operation mode circuit analysis identified the capability of extracting partial product outputs from the multiplier, intermediate shift register values, and even internal constants used by the circuit to perform Boolean logic operations. These behaviors are not strictly malicious or illegal; however, they are clear examples of real functionality not being fully disclosed by the vendor. See Section 5.2 for additional information on these uncovered undocumented modes.

Figure 3.1 shows the overall user's guide level circuit of the DSP48E. The introduction of undocumented states comes from three contributors, the Cascade Circuit, determined by A and B inputs, the Arithmetic Logic Unit, determined by the ALUMODE input, and the Operation Mode Logic, determined by the OPMODE input. For this study, the ALUMODE and OPMODE inputs have been evaluated together since the ALU logic operations are effected by the inputs provided as part of the Operation mode selection.

### 5.1 Cascade Circuit Results

The Cascade Circuit was first identified to have five undocumented modes for both A and B inputs. On-chip Circuit Analysis concluded that the four valid modes perform the expected operation. For the undocumented

Table 5.1: Identified undocumented modes using knowledge based partitioning and the resulting on-chip analysis of undocumented modes totaling 74.8% coverage

	Knowledge Based Partitioning Identified Undocumented Modes	On-Chip Circuit Analysis Evaluated Undocumented Modes
<b>Cascade Register:</b>		
A Input sub-circuit	5	7
B Input sub-circuit	5	7
<b>Op/ALU Modes:</b>		
X Operand Multiplexer	268	268
Y Operand Multiplexer	512	512
Z Operand Multiplexer	728	344
Register Output	0	0
Pattern Detection Logic	0	0
Multiplier	0	0
<b>Total:</b>	<b>1518</b>	<b>1136</b>

Table 5.2: Cascade Mode Undocumented Mode Testing Summary (full details in Appendix A)

Knowledge Based Partitioning	On-Chip Circuit Analysis	
	Identified Undocumented Modes	Evaluated Undocumented Modes
Table 8.1	5	7
Table 8.2	5	7
<b>Total</b>	<b>10</b>	<b>14</b>

modes, only one mode matched the initial behavioral model's expected functionality. That is to say, based on the cascade circuits documentation provided by the vendor, the expected behavior that was modeled as a result of the knowledge based partitioning was incomplete. To create a complete empirical model, on-chip testing was required and was able to provide complete coverage of the undocumented functionality. This fact emphasizes a limitation that the vendor documentation does not provide accurate circuit descriptions. While the vendor tools may catch some of these undocumented modes and prevent a full design from building; this study has further shown it is possible to manipulate the tools and the design flow to enter these undocumented modes.

Furthermore, ISI's developed techniques to analyze the bitstreams used during configuration identified two additional states that were not covered through the REG and CASC register settings. This results in seven undocumented modes for the cascade circuit instead of the originally calculated five that was based on the vendor supplied user guide documentation.

The two additional modes were uncovered by generating a complete list of all possible cascade modes through XDL configuration, then analyzing the bits in the bitstream that control each mode. It was discovered that three bits control the cascade registers, yet of the  $2^3 = 8$  possible bitstream configurations, '110' and '111' were not generated (see Tables 8.1 and 8.2 in Appendix A for a full listing of all bitstream configurations). The resulting undocumented functionality produce one previously unobserved (and undocumented) behavior of AREG/BREG being able to be bypassed while ACASREG/BCASREG could be enabled. The second undocumented behavior ended up replicating a previously observed mode of both AERG/BREG and ACASCREG/BCASCREG being bypassed.

In this example, the actual behavior is interesting, but moreover the fact that the techniques could discover the undocumented functionality indicates the approach is capable of quickly analyzing and identifying supplemental undocumented modes in other IP blocks. A summary of the cascade circuit testing can be found in Table 5.2, highlighting the number of undocumented modes evaluated during this study. Additional information on these modes and their undocumented behavior are listed in Appendix A.

## 5.2 ALU and Operation Mode Results

In addition to the cascade circuit the ALU and Operation mode run-time settings were evaluated. The Appendix has the full listing of all undocumented modes and those that were tested under this study. The light blue row markings highlight the Op Mode and ALU Modes that are undocumented. All rows present the observed outputs for these modes that were evaluated on real hardware during the run-time on-chip testing.

Due to the run-time nature of the ALU and Operation modes it was possible to quickly evaluate undocumented modes and determine the resulting functionality. While the Appendix lists out in great detail the undocumented functionality for each of these modes, this study was able to identify and extract partial product outputs from the multiplier, intermediate shift register values, and internal constants used by the circuit to perform Boolean logic operations. Using these undocumented modes it maybe possible to further extract details of the multiplier and two-stage adder/subtractor circuits, far beyond what is provided by the vendor's documentation.

Only ALU Modes *1001,1010,1011* were not evaluated during this testing. While all combinations of Op Modes

Table 5.3: ALU and Op Mode Undocumented Mode Testing Summary (full details in Appendix B)

Table Reference	ALU Mode[3:0]	Undocumented Modes	Undocumented Modes Evaluated
Table 8.4	0000	67	67
Table 8.5	0001	67	67
Table 8.6	0010	67	67
Table 8.7	0011	67	67
Table 8.8	0100	91	91
Table 8.9	0101	91	91
Table 8.10	0110	91	91
Table 8.11	0111	91	91
Table 8.12	1000	128	128
Table 8.13	1001	128	0
Table 8.14	1010	128	0
Table 8.15	1011	128	0
Table 8.16	1100	91	91
Table 8.17	1101	91	91
Table 8.18	1110	91	91
Table 8.19	1111	91	91
<b>Total</b>	—	<b>1508</b>	<b>1124</b>

with these ALU modes are undocumented, this study did not dive into these at present. The techniques developed thus far could be further extended to cover the remaining modes, as well as, be applied to other Hard IP blocks to provide greater device coverage. Table 5.3 provides a summary of the ALU and Op mode testing results, highlighting for each ALU mode the number of undocumented modes that were identify and evaluated as part of this study.

## 6 | Conclusion

In summary, this study proved very successful on a number of fronts. This is the first known study to utilize on-chip testing to validate the discovery of undocumented features. This approach proved to be more thorough than manual analysis and patent review as the discovery of additional modes in the Cascade circuit through bitstream injection revealed. One of the main focuses of our research here, was to also provide functional descriptions of what the circuit is doing. Many previous efforts merely provided a gate level description of undocumented functionality, but gave no indication to the end consumer of the data as to if the functionality was benign or malicious. In the appendix, we are able to clearly provide the mathematical behavior of all 1,136 evaluated undocumented modes.

It is important to note that the approach here is well tailored for FPGA devices, where user documentation is often provided at an abstracted circuit level. We believe this research is an important first step in effective approaches for discovery of undocumented functionality. For FPGAs, future research can further explore the scalability of this approach as other, larger and more complicated pieces of IP can be explored. This approach may even be viable for other processor types, where additional inference steps can address the translation of the even higher level documentation provided for general purpose processors into behavioral level functionality.



## 7 | References

- [1] C. W. S. Skorobogatov, "Breakthrough silicon scanning discovers backdoor in military chip," *Cryptographic Hardware and Embedded Systems Workshop*, 9 2012.
- [2] D. Kaufman, "An analytical framework for cyber security," 2012.
- [3] Xilinx, Inc., "Virtex-5 FPGA XtremeDSP Design Considerations User Guide (UG193) v3.5," January 2012.
- [4] Xilinx, Inc., "ML505/ML506/ML507 LE5v0a7luEatviaolnuaPtliatnform Platform User Guide (UG347) v3.1.2," May 2011.

## 8 | Appendix

### A Cascade Circuit Full Results

The DSP48's A and B cascade circuits were first analyzed and ten modes were initially determined to be undocumented. The run-time on-chip testing performed by USC/ISI validated the outputs as follows. The white rows indicate documented, valid modes from the vendor documentation. The observed outputs during run-time testing matched the documented and expected behaviors. The light blue highlighted rows indicate modes that are documentation. For these modes the observed output is recorded based on the run-time on-chip testing. Finally, the red rows (index modes 9 and 10) have been found through bitstream manipulation and their corresponding on-chip testing observed outputs are reported. As a result, a total of 14 undocumented modes, for both the A and B cascade circuits, have been tested and their corresponding observed outputs have been reported, shown in Tables 8.1 and 8.2.

Table 8.1: A Cascade Register Observed Results

Index	Bitstream Configuration[2:0]	AREG Observed Output	ACAS Observed Output
0	101	0	0
1	001	1	1
2	010	2	1
3	011	2	2
4	000	1	0
5	100	0	0
6	100	0	0
7	000	1	0
8	010	2	1
9	110	0	1
10	111	0	0

Table 8.2: B Cascade Register Observed Results

Index	Bitstream Configuration[2:0]	BREG Observed Output	BCAS Observed Output
0	101	0	0
1	001	1	1
2	010	2	1
3	011	2	2
4	000	1	0
5	100	0	0
6	100	0	0
7	000	1	0
8	010	2	1
9	110	0	1
10	111	0	0

### B ALU Op Mode Full Results

The evaluated ALU and Op Modes tested with USC/ISI's techniques. Table 8.3 provides a description of all of the terms used throughout Tables 8.4-8.19. Each table represents an specific ALU Mode setting, while each row in the table represents a specific Op Mode for the given ALU mode. The right column represents what the observed outputs are for each given mode from run-time on-chip testing. All output cells in white represent valid modes for

Table 8.3: Terms and descriptions used in ALUMODE Tables 8.4-8.19

Term	Description
PP1	Multiplier partial product 1
PP2	Multiplier partial product 2
P	Data output from second stage ALU
A:B	30-bit A and 18-bit B inputs concatenated together to second stage of ALU
C	48-bit data input to second stage of ALU
RS_PCIN	Cascaded data input from PCOUT of previous DSP48E shifted right 17-bits
RS_P	P shifted right 17-bits
0	48-bit vector of 0's
48'FFFFFFFFFFFFFF	48-bit vector of 1's
+	ALU addition
-	ALU subtraction
*	Multiplication
$\oplus$	Logic XOR
$\wedge$	Logic AND
$\vee$	Logic OR
$\neg$	Logic NOT

the given ALU and Op mode settings, from the documentation. The value in the cell represents the observed or expected output. Undocumented modes not specified by the documentation or explicitly stated as illegal modes have their cells highlighted in light blue. The value of the cell represents what was determined as the functionality based on the knowledge-based partitioning and on-chip testing.

Table 8.4: ALUMODE 0000 Observed Results

OP Modes							Observed Outputs	
Z	Y			X				
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	PP1
0	0	0	0	0	0	1	0	P
0	0	0	0	0	0	1	1	A : B
0	0	0	0	0	1	0	0	PP2
0	0	0	0	0	1	0	1	PP1 + PP2
0	0	0	0	0	1	1	0	P + PP2
0	0	0	0	0	1	1	1	A : B + PP2
0	0	0	0	1	0	0	0	48'FFFFFFFFFFFFFF
0	0	0	0	1	0	0	1	PP1 + 48'FFFFFFFFFFFFFF
0	0	0	0	1	0	1	0	P + 48'FFFFFFFFFFFFFF
0	0	0	0	1	0	1	1	A : B + 48'FFFFFFFFFFFFFF
0	0	0	0	1	1	0	0	C
0	0	0	0	1	1	0	1	PP1 + C
0	0	0	0	1	1	1	0	P + C
0	0	0	0	1	1	1	1	A : B + C
0	0	0	1	0	0	0	0	0 + PCIN
0	0	0	1	0	0	0	1	PP1 + PCIN
0	0	0	1	0	0	1	0	P + PCIN
0	0	0	1	0	0	1	1	A : B + PCIN
0	0	0	1	0	1	0	0	PP2 + PCIN
0	0	0	1	0	1	0	1	PP1 + PP2 + PCIN
0	0	0	1	0	1	1	0	P + PP2 + PCIN
0	0	0	1	0	1	1	1	A : B + PP2 + PCIN
0	0	0	1	1	0	0	0	48'FFFFFFFFFFFFFF + PCIN
0	0	0	1	1	0	0	1	PP1 + 48'FFFFFFFFFFFFFF + PCIN
0	0	0	1	1	0	1	0	P + 48'FFFFFFFFFFFFFF + PCIN
0	0	0	1	1	0	1	1	A : B + 48'FFFFFFFFFFFFFF + PCIN

Continued on next page

Table 8.4: ALUMODE 0000 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	0	1	1	1	0	0	$C + PCIN$
0	0	0	1	1	1	0	$PP1 + C + PCIN$
0	0	0	1	1	1	1	$P + C + PCIN$
0	0	1	1	1	1	1	$A : B + C + PCIN$
0	1	0	0	0	0	0	$0 + P$
0	1	0	0	0	0	1	$PP1 + P$
0	1	0	0	0	1	0	$P + P$
0	1	0	0	0	1	1	$A : B + P$
0	1	0	0	1	0	0	$PP2 + P$
0	1	0	0	1	0	1	$PP1 + PP2 + P$
0	1	0	0	1	1	0	$P + PP2 + P$
0	1	0	0	1	1	1	$A : B + PP2 + P$
0	1	0	1	0	0	0	$48'FFFFFFFFFFFFFF + P$
0	1	0	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + P$
0	1	0	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + P$
0	1	0	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + P$
0	1	0	1	1	0	0	$C + P$
0	1	0	1	1	0	1	$PP1 + C + P$
0	1	0	1	1	1	0	$P + C + P$
0	1	0	1	1	1	1	$A : B + C + P$
0	1	1	0	0	0	0	$0 + C$
0	1	1	0	0	0	1	$PP1 + C$
0	1	1	0	0	1	0	$P + C$
0	1	1	0	0	1	1	$A : B + C$
0	1	1	0	1	0	0	$PP2 + C$
0	1	1	0	1	0	1	$PP1 + PP2 + C$
0	1	1	0	1	1	0	$P + PP2 + C$
0	1	1	0	1	1	1	$A : B + PP2 + C$
0	1	1	1	0	0	0	$48'FFFFFFFFFFFFFF + C$
0	1	1	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + C$
0	1	1	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + C$
0	1	1	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + C$
0	1	1	1	1	0	0	$C + C$
0	1	1	1	1	0	1	$PP1 + C + C$
0	1	1	1	1	1	0	$P + C + C$
0	1	1	1	1	1	1	$A : B + C + C$
1	0	0	0	0	0	0	$0 + P$
1	0	0	0	0	0	1	$PP1 + P$
1	0	0	0	0	1	0	$P + P$
1	0	0	0	0	1	1	$A : B + P$
1	0	0	0	1	0	0	$PP2 + P$
1	0	0	0	1	0	1	$PP1 + PP2 + P$
1	0	0	0	1	1	0	$P + PP2 + P$
1	0	0	0	1	1	1	$A : B + PP2 + P$
1	0	0	1	0	0	0	$48'FFFFFFFFFFFFFF + P$
1	0	0	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + P$
1	0	0	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + P$
1	0	0	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + P$
1	0	0	1	1	0	0	$C + P$
1	0	0	1	1	0	1	$PP1 + C + P$
1	0	0	1	1	1	0	$P + C + P$
1	0	0	1	1	1	1	$A : B + C + P$
1	0	1	0	0	0	0	$0 + RS\_PCIN$
1	0	1	0	0	0	1	$PP1 + RS\_PCIN$
1	0	1	0	0	1	0	$P + RS\_PCIN$
1	0	1	0	0	1	1	$A : B + RS\_PCIN$
1	0	1	0	1	0	0	$PP2 + RS\_PCIN$
1	0	1	0	1	0	1	$PP1 + PP2 + RS\_PCIN$
1	0	1	0	1	1	0	$P + PP2 + RS\_PCIN$
1	0	1	0	1	1	1	$A : B + PP2 + RS\_PCIN$
1	0	1	1	0	0	0	$48'FFFFFFFFFFFFFF + RS\_PCIN$

Continued on next page

Table 8.4: ALUMODE 0000 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	0	1	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + RS\_PCIN$
1	0	1	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + RS\_PCIN$
1	0	1	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + RS\_PCIN$
1	0	1	1	1	0	0	$C + RS\_PCIN$
1	0	1	1	1	0	1	$PP1 + C + RS\_PCIN$
1	0	1	1	1	1	0	$P + C + RS\_PCIN$
1	0	1	1	1	1	1	$A : B + C + RS\_PCIN$
1	1	0	0	0	0	0	$0 + RS\_P$
1	1	0	0	0	0	1	$PP1 + RS\_P$
1	1	0	0	0	1	0	$P + RS\_P$
1	1	0	0	0	1	1	$A : B + RS\_P$
1	1	0	0	1	0	0	$PP2 + RS\_P$
1	1	0	0	1	0	1	$PP1 + PP2 + RS\_P$
1	1	0	0	1	1	0	$P + PP2 + RS\_P$
1	1	0	0	1	1	1	$A : B + PP2 + RS\_P$
1	1	0	1	0	0	0	$48'FFFFFFFFFFFFFF + RS\_P$
1	1	0	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	0	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	0	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	0	1	1	0	0	$C + RS\_P$
1	1	0	1	1	0	1	$PP1 + C + RS\_P$
1	1	0	1	1	1	0	$P + C + RS\_P$
1	1	0	1	1	1	1	$A : B + C + RS\_P$
1	1	1	0	0	0	0	$0 + RS\_P$
1	1	1	0	0	0	1	$PP1 + RS\_P$
1	1	1	0	0	1	0	$P + RS\_P$
1	1	1	0	0	1	1	$A : B + RS\_P$
1	1	1	0	1	0	0	$PP2 + RS\_P$
1	1	1	0	1	0	1	$PP1 + PP2 + RS\_P$
1	1	1	0	1	1	0	$P + PP2 + RS\_P$
1	1	1	0	1	1	1	$A : B + PP2 + RS\_P$
1	1	1	1	0	0	0	$48'FFFFFFFFFFFFFF + RS\_P$
1	1	1	1	0	0	1	$PP1 + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	1	1	0	1	0	$P + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	1	1	0	1	1	$A : B + 48'FFFFFFFFFFFFFF + RS\_P$
1	1	1	1	1	0	0	$C + RS\_P$
1	1	1	1	1	0	1	$PP1 + C + RS\_P$
1	1	1	1	1	1	0	$P + C + RS\_P$
1	1	1	1	1	1	1	$A : B + C + RS\_P$

Table 8.5: ALUMODE 0001 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$-(0) - 1$
0	0	0	0	0	0	1	$-(PP1) - 1$
0	0	0	0	0	1	0	$-(P) - 1$
0	0	0	0	0	1	1	$-(A : B) - 1$
0	0	0	0	1	0	0	$-(PP2) - 1$
0	0	0	0	1	0	1	$-(PP1 + PP2) - 1$
0	0	0	0	1	1	0	$-(P + PP2) - 1$
0	0	0	0	1	1	1	$-(A : B + PP2) - 1$
0	0	0	1	0	0	0	$-(48'FFFFFFFFFFFFFFFF) - 1$
0	0	0	1	0	0	1	$-(PP1 + 48'FFFFFFFFFFFFFFFF) - 1$
0	0	0	1	0	1	0	$-(P + 48'FFFFFFFFFFFFFFFF) - 1$
0	0	0	1	0	1	1	$-(A : B + 48'FFFFFFFFFFFFFFFF) - 1$
0	0	0	1	1	0	0	$-(C) - 1$

Continued on next page

Table 8.5: ALUMODE 0001 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	1	1	0	1	$-(PP1 + C) - 1$
0	0	0	1	1	1	0	$-(P + C) - 1$
0	0	0	1	1	1	1	$-(A : B + C) - 1$
0	0	1	0	0	0	0	$-PCIN + (0) - 1$
0	0	1	0	0	0	1	$-PCIN + (PP1) - 1$
0	0	1	0	0	1	0	$-PCIN + (P) - 1$
0	0	1	0	0	1	1	$-PCIN + (A : B) - 1$
0	0	1	0	1	0	0	$-PCIN + (PP2) - 1$
0	0	1	0	1	0	1	$-PCIN + (PP1 + PP2) - 1$
0	0	1	0	1	1	0	$-PCIN + (P + PP2) - 1$
0	0	1	0	1	1	1	$-PCIN + (A : B + PP2) - 1$
0	0	1	1	0	0	0	$-PCIN + (48'FFFFFFFFFFFFFF) - 1$
0	0	1	1	0	0	1	$-PCIN + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
0	0	1	1	0	1	0	$-PCIN + (P + 48'FFFFFFFFFFFFFF) - 1$
0	0	1	1	0	1	1	$-PCIN + (A : B + 48'FFFFFFFFFFFFFF) - 1$
0	0	1	1	1	0	0	$-PCIN + (C) - 1$
0	0	1	1	1	1	0	$-PCIN + (PP1 + C) - 1$
0	0	1	1	1	1	1	$-PCIN + (P + C) - 1$
0	0	1	1	1	1	1	$-PCIN + (A : B + C) - 1$
0	1	0	0	0	0	0	$-P + (0) - 1$
0	1	0	0	0	0	1	$-P + (PP1) - 1$
0	1	0	0	0	1	0	$-P + (P) - 1$
0	1	0	0	0	1	1	$-P + (A : B) - 1$
0	1	0	0	1	0	0	$-P + (PP2) - 1$
0	1	0	0	1	0	1	$-P + (PP1 + PP2) - 1$
0	1	0	0	1	1	0	$-P + (P + PP2) - 1$
0	1	0	0	1	1	1	$-P + (A : B + PP2) - 1$
0	1	0	1	0	0	0	$-P + (48'FFFFFFFFFFFFFF) - 1$
0	1	0	1	0	0	1	$-P + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
0	1	0	1	0	1	0	$-P + (P + 48'FFFFFFFFFFFFFF) - 1$
0	1	0	1	0	1	1	$-P + (A : B + 48'FFFFFFFFFFFFFF) - 1$
0	1	0	1	1	0	0	$-P + (C) - 1$
0	1	0	1	1	1	0	$-P + (PP1 + C) - 1$
0	1	0	1	1	1	1	$-P + (P + C) - 1$
0	1	0	1	1	1	1	$-P + (A : B + C) - 1$
0	1	1	0	0	0	0	$-C + (0) - 1$
0	1	1	0	0	0	1	$-C + (PP1) - 1$
0	1	1	0	0	1	0	$-C + (P) - 1$
0	1	1	0	0	1	1	$-C + (A : B) - 1$
0	1	1	0	1	0	0	$-C + (PP2) - 1$
0	1	1	0	1	0	1	$-C + (PP1 + PP2) - 1$
0	1	1	0	1	1	0	$-C + (P + PP2) - 1$
0	1	1	0	1	1	1	$-C + (A : B + PP2) - 1$
0	1	1	1	0	0	0	$-C + (48'FFFFFFFFFFFFFF) - 1$
0	1	1	1	0	0	1	$-C + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
0	1	1	1	0	1	0	$-C + (P + 48'FFFFFFFFFFFFFF) - 1$
0	1	1	1	0	1	1	$-C + (A : B + 48'FFFFFFFFFFFFFF) - 1$
0	1	1	1	1	0	0	$-C + (C) - 1$
0	1	1	1	1	0	1	$-C + (PP1 + C) - 1$
0	1	1	1	1	1	0	$-C + (P + C) - 1$
0	1	1	1	1	1	1	$-C + (A : B + C) - 1$
1	0	0	0	0	0	0	$-P + (0) - 1$
1	0	0	0	0	0	1	$-P + (PP1) - 1$
1	0	0	0	0	1	0	$-P + (P) - 1$
1	0	0	0	0	1	1	$-P + (A : B) - 1$
1	0	0	0	1	0	0	$-P + (PP2) - 1$
1	0	0	0	1	0	1	$-P + (PP1 + PP2) - 1$
1	0	0	0	1	1	0	$-P + (P + PP2) - 1$
1	0	0	0	1	1	1	$-P + (A : B + PP2) - 1$
1	0	0	1	0	0	0	$-P + (48'FFFFFFFFFFFFFF) - 1$
1	0	0	1	0	0	1	$-P + (PP1 + 48'FFFFFFFFFFFFFF) - 1$

Continued on next page

Table 8.5: ALUMODE 0001 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	0	0	1	0	1	0	$-P + (P + 48'FFFFFFFFFFFFFF) - 1$
1	0	0	1	0	1	1	$-P + (A : B + 48'FFFFFFFFFFFFFF) - 1$
1	0	0	1	1	0	0	$-P + (C) - 1$
1	0	0	1	1	0	1	$-P + (PP1 + C) - 1$
1	0	0	1	1	1	0	$-P + (P + C) - 1$
1	0	0	1	1	1	1	$-P + (A : B + C) - 1$
1	0	1	0	0	0	0	$-RS\_PCIN + (0) - 1$
1	0	1	0	0	0	1	$-RS\_PCIN + (PP1) - 1$
1	0	1	0	0	1	0	$-RS\_PCIN + (P) - 1$
1	0	1	0	0	1	1	$-RS\_PCIN + (A : B) - 1$
1	0	1	0	1	0	0	$-RS\_PCIN + (PP2) - 1$
1	0	1	0	1	0	1	$-RS\_PCIN + (PP1 + PP2) - 1$
1	0	1	0	1	1	0	$-RS\_PCIN + (P + PP2) - 1$
1	0	1	0	1	1	1	$-RS\_PCIN + (A : B + PP2) - 1$
1	0	1	1	0	0	0	$-RS\_PCIN + (48'FFFFFFFFFFFFFF) - 1$
1	0	1	1	0	0	1	$-RS\_PCIN + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
1	0	1	1	0	1	0	$-RS\_PCIN + (P + 48'FFFFFFFFFFFFFF) - 1$
1	0	1	1	0	1	1	$-RS\_PCIN + (A : B + 48'FFFFFFFFFFFFFF) - 1$
1	0	1	1	1	0	0	$-RS\_PCIN + (C) - 1$
1	0	1	1	1	0	1	$-RS\_PCIN + (PP1 + C) - 1$
1	0	1	1	1	1	0	$-RS\_PCIN + (P + C) - 1$
1	0	1	1	1	1	1	$-RS\_PCIN + (A : B + C) - 1$
1	1	0	0	0	0	0	$-RS\_P + (0) - 1$
1	1	0	0	0	0	1	$-RS\_P + (PP1) - 1$
1	1	0	0	0	1	0	$-RS\_P + (P) - 1$
1	1	0	0	0	1	1	$-RS\_P + (A : B) - 1$
1	1	0	0	1	0	0	$-RS\_P + (PP2) - 1$
1	1	0	0	1	0	1	$-RS\_P + (PP1 + PP2) - 1$
1	1	0	0	1	1	0	$-RS\_P + (P + PP2) - 1$
1	1	0	0	1	1	1	$-RS\_P + (A : B + PP2) - 1$
1	1	0	1	0	0	0	$-RS\_P + (48'FFFFFFFFFFFFFF) - 1$
1	1	0	1	0	0	1	$-RS\_P + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
1	1	0	1	0	1	0	$-RS\_P + (P + 48'FFFFFFFFFFFFFF) - 1$
1	1	0	1	0	1	1	$-RS\_P + (A : B + 48'FFFFFFFFFFFFFF) - 1$
1	1	0	1	1	0	0	$-RS\_P + (C) - 1$
1	1	0	1	1	0	1	$-RS\_P + (PP1 + C) - 1$
1	1	0	1	1	1	0	$-RS\_P + (P + C) - 1$
1	1	0	1	1	1	1	$-RS\_P + (A : B + C) - 1$
1	1	1	0	0	0	0	$-RS\_P + (0) - 1$
1	1	1	0	0	0	1	$-RS\_P + (PP1) - 1$
1	1	1	0	0	1	0	$-RS\_P + (P) - 1$
1	1	1	0	0	1	1	$-RS\_P + (A : B) - 1$
1	1	1	0	1	0	0	$-RS\_P + (PP2) - 1$
1	1	1	0	1	0	1	$-RS\_P + (PP1 + PP2) - 1$
1	1	1	0	1	1	0	$-RS\_P + (P + PP2) - 1$
1	1	1	0	1	1	1	$-RS\_P + (A : B + PP2) - 1$
1	1	1	1	0	0	0	$-RS\_P + (48'FFFFFFFFFFFFFF) - 1$
1	1	1	1	0	0	1	$-RS\_P + (PP1 + 48'FFFFFFFFFFFFFF) - 1$
1	1	1	1	0	1	0	$-RS\_P + (P + 48'FFFFFFFFFFFFFF) - 1$
1	1	1	1	0	1	1	$-RS\_P + (A : B + 48'FFFFFFFFFFFFFF) - 1$
1	1	1	1	1	0	0	$-RS\_P + (C) - 1$
1	1	1	1	1	0	1	$-RS\_P + (PP1 + C) - 1$
1	1	1	1	1	1	0	$-RS\_P + (P + C) - 1$
1	1	1	1	1	1	1	$-RS\_P + (A : B + C) - 1$

Table 8.6: ALUMODE 0010 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	-0-0-0-0-1
0	0	0	0	0	0	1	-0-PP1-0-0-1
0	0	0	0	0	1	0	-0-P-0-0-1
0	0	0	0	0	1	1	-0-A:B-0-0-1
0	0	0	0	1	0	0	-0-0-PP2-0-1
0	0	0	0	1	0	1	-0-PP1-PP2-0-1
0	0	0	0	1	1	0	-0-P-PP2-0-1
0	0	0	0	1	1	1	-0-A:B-PP2-0-1
0	0	0	1	0	0	0	-0-0-48'FFFFFFFFFFFFFF-0-1
0	0	0	1	0	0	1	-0-PP1-48'FFFFFFFFFFFFFF-0-1
0	0	0	1	0	1	0	-0-P-48'FFFFFFFFFFFFFF-0-1
0	0	0	1	0	1	1	-0-A:B-48'FFFFFFFFFFFFFF-0-1
0	0	0	1	1	0	0	-0-0-C-0-1
0	0	0	1	1	0	1	-0-PP1-C-0-1
0	0	0	1	1	1	0	-0-P-C-0-1
0	0	0	1	1	1	1	-0-A:B-C-0-1
0	0	1	0	0	0	0	-PCIN-0-0-0-0-1
0	0	1	0	0	0	1	-PCIN-PP1-0-0-0-1
0	0	1	0	0	1	0	-PCIN-P-0-0-0-1
0	0	1	0	0	1	1	-PCIN-A:B-0-0-0-1
0	0	1	0	1	0	0	-PCIN-0-PP2-0-1
0	0	1	0	1	0	1	-PCIN-PP1-PP2-0-1
0	0	1	0	1	1	0	-PCIN-P-PP2-0-1
0	0	1	0	1	1	1	-PCIN-A:B-PP2-0-1
0	0	1	1	0	0	0	-PCIN-0-48'FFFFFFFFFFFFFF-0-1
0	0	1	1	0	0	1	-PCIN-PP1-48'FFFFFFFFFFFFFF-0-1
0	0	1	1	0	1	0	-PCIN-P-48'FFFFFFFFFFFFFF-0-1
0	0	1	1	0	1	1	-PCIN-A:B-48'FFFFFFFFFFFFFF-0-1
0	0	1	1	1	0	0	-PCIN-0-C-0-1
0	0	1	1	1	0	1	-PCIN-PP1-C-0-1
0	0	1	1	1	1	0	-PCIN-P-C-0-1
0	0	1	1	1	1	1	-PCIN-A:B-C-0-1
0	1	0	0	0	0	0	-P-0-0-0-0-1
0	1	0	0	0	0	1	-P-PP1-0-0-0-1
0	1	0	0	0	1	0	-P-P-0-0-0-1
0	1	0	0	0	1	1	-P-A:B-0-0-0-1
0	1	0	0	1	0	0	-P-0-PP2-0-1
0	1	0	0	1	0	1	-P-PP1-PP2-0-1
0	1	0	0	1	1	0	-P-P-PP2-0-1
0	1	0	0	1	1	1	-P-A:B-PP2-0-1
0	1	0	1	0	0	0	-P-0-48'FFFFFFFFFFFFFF-0-1
0	1	0	1	0	0	1	-P-PP1-48'FFFFFFFFFFFFFF-0-1
0	1	0	1	0	1	0	-P-P-48'FFFFFFFFFFFFFF-0-1
0	1	0	1	0	1	1	-P-A:B-48'FFFFFFFFFFFFFF-0-1
0	1	0	1	1	0	0	-P-0-C-0-1
0	1	0	1	1	0	1	-P-PP1-C-0-1
0	1	0	1	1	1	0	-P-P-C-0-1
0	1	0	1	1	1	1	-P-A:B-C-0-1
0	1	1	0	0	0	0	-C-0-0-0-0-1
0	1	1	0	0	0	1	-C-PP1-0-0-0-1
0	1	1	0	0	1	0	-C-P-0-0-0-1
0	1	1	0	0	1	1	-C-A:B-0-0-0-1
0	1	1	0	1	0	0	-C-0-PP2-0-1
0	1	1	0	1	0	1	-C-PP1-PP2-0-1
0	1	1	0	1	1	0	-C-P-PP2-0-1
0	1	1	0	1	1	1	-C-A:B-PP2-0-1
0	1	1	1	0	0	0	-C-0-48'FFFFFFFFFFFFFF-0-1
0	1	1	1	0	0	1	-C-PP1-48'FFFFFFFFFFFFFF-0-1
0	1	1	1	0	1	0	-C-P-48'FFFFFFFFFFFFFF-0-1
0	1	1	1	0	1	1	-C-A:B-48'FFFFFFFFFFFFFF-0-1
0	1	1	1	1	0	0	-C-0-C-0-1

Continued on next page



Table 8.6: ALUMODE 0010 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	1	1	1	1	0	1	$-C - PP1 - C - 0 - 1$
0	1	1	1	1	1	0	$-C - P - C - 0 - 1$
0	1	1	1	1	1	1	$-C - A : B - C - 0 - 1$
1	0	0	0	0	0	0	$-P - 0 - 0 - 0 - 1$
1	0	0	0	0	0	1	$-P - PP1 - 0 - 0 - 1$
1	0	0	0	0	1	0	$-P - P - 0 - 0 - 1$
1	0	0	0	0	1	1	$-P - A : B - 0 - 0 - 1$
1	0	0	0	1	0	0	$-P - 0 - PP2 - 0 - 1$
1	0	0	0	1	0	1	$-P - PP1 - PP2 - 0 - 1$
1	0	0	0	1	1	0	$-P - P - PP2 - 0 - 1$
1	0	0	0	1	1	1	$-P - A : B - PP2 - 0 - 1$
1	0	0	1	0	0	0	$-P - 0 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	0	1	$-P - PP1 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	1	0	$-P - P - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	1	1	$-P - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	1	0	0	$-P - 0 - C - 0 - 1$
1	0	0	1	1	0	1	$-P - PP1 - C - 0 - 1$
1	0	0	1	1	1	0	$-P - P - C - 0 - 1$
1	0	0	1	1	1	1	$-P - A : B - C - 0 - 1$
1	0	1	0	0	0	0	$-RS\_PCIN - 0 - 0 - 0 - 1$
1	0	1	0	0	0	1	$-RS\_PCIN - PP1 - 0 - 0 - 1$
1	0	1	0	0	1	0	$-RS\_PCIN - P - 0 - 0 - 1$
1	0	1	0	0	1	1	$-RS\_PCIN - A : B - 0 - 0 - 1$
1	0	1	0	1	0	0	$-RS\_PCIN - 0 - PP2 - 0 - 1$
1	0	1	0	1	0	1	$-RS\_PCIN - PP1 - PP2 - 0 - 1$
1	0	1	0	1	1	0	$-RS\_PCIN - P - PP2 - 0 - 1$
1	0	1	0	1	1	1	$-RS\_PCIN - A : B - PP2 - 0 - 1$
1	0	1	1	0	0	0	$-RS\_PCIN - 0 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	0	1	$-RS\_PCIN - PP1 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	1	0	$-RS\_PCIN - P - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	1	1	$-RS\_PCIN - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	1	0	0	$-RS\_PCIN - 0 - C - 0 - 1$
1	0	1	1	1	0	1	$-RS\_PCIN - PP1 - C - 0 - 1$
1	0	1	1	1	1	0	$-RS\_PCIN - P - C - 0 - 1$
1	0	1	1	1	1	1	$-RS\_PCIN - A : B - C - 0 - 1$
1	1	0	0	0	0	0	$-RS\_P - 0 - 0 - 0 - 1$
1	1	0	0	0	0	1	$-RS\_P - PP1 - 0 - 0 - 1$
1	1	0	0	0	1	0	$-RS\_P - P - 0 - 0 - 1$
1	1	0	0	0	1	1	$-RS\_P - A : B - 0 - 0 - 1$
1	1	0	0	1	0	0	$-RS\_P - 0 - PP2 - 0 - 1$
1	1	0	0	1	0	1	$-RS\_P - PP1 - PP2 - 0 - 1$
1	1	0	0	1	1	0	$-RS\_P - P - PP2 - 0 - 1$
1	1	0	0	1	1	1	$-RS\_P - A : B - PP2 - 0 - 1$
1	1	0	1	0	0	0	$-RS\_P - 0 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	0	0	1	$-RS\_P - PP1 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	0	1	0	$-RS\_P - P - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	0	1	1	$-RS\_P - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	1	0	0	$-RS\_P - 0 - C - 0 - 1$
1	1	0	1	1	0	1	$-RS\_P - PP1 - C - 0 - 1$
1	1	0	1	1	1	0	$-RS\_P - P - C - 0 - 1$
1	1	0	1	1	1	1	$-RS\_P - A : B - C - 0 - 1$
1	1	1	0	0	0	0	$-RS\_P - 0 - 0 - 0 - 1$
1	1	1	0	0	0	1	$-RS\_P - PP1 - 0 - 0 - 1$
1	1	1	0	0	1	0	$-RS\_P - P - 0 - 0 - 1$
1	1	1	0	0	1	1	$-RS\_P - A : B - 0 - 0 - 1$
1	1	1	0	1	0	0	$-RS\_P - 0 - PP2 - 0 - 1$
1	1	1	0	1	0	1	$-RS\_P - PP1 - PP2 - 0 - 1$
1	1	1	0	1	1	0	$-RS\_P - P - PP2 - 0 - 1$
1	1	1	0	1	1	1	$-RS\_P - A : B - PP2 - 0 - 1$
1	1	1	1	0	0	0	$-RS\_P - 0 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	0	0	1	$-RS\_P - PP1 - 48'FFFFFFFFFFFFFF - 0 - 1$

Continued on next page

Table 8.6: ALUMODE 0010 Observed Results (cont.)

OP Modes							Observed Outputs
Z			Y		X		
1	1	1	1	0	1	0	$-RS\_P - P - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	0	1	1	$-RS\_P - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	1	0	0	$-RS\_P - 0 - C - 0 - 1$
1	1	1	1	1	0	1	$-RS\_P - PP1 - C - 0 - 1$
1	1	1	1	1	1	0	$-RS\_P - P - C - 0 - 1$
1	1	1	1	1	1	1	$-RS\_P - A : B - C - 0 - 1$

Table 8.7: ALUMODE 0011 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$-0-0-0-0-1$
0	0	0	0	0	0	1	$-0-PP1-0-0-1$
0	0	0	0	0	1	0	$-0-P-0-0-1$
0	0	0	0	0	1	1	$-0-A:B-0-0-1$
0	0	0	0	1	0	0	$-0-0-PP2-0-1$
0	0	0	0	1	0	1	$-0-PP1-PP2-0-1$
0	0	0	0	1	1	0	$-0-P-PP2-0-1$
0	0	0	0	1	1	1	$-0-A:B-PP2-0-1$
0	0	0	1	0	0	0	$-0-0-48'FFFFFFFFFFFFFF-0-1$
0	0	0	1	0	0	1	$-0-PP1-48'FFFFFFFFFFFFFF-0-1$
0	0	0	1	0	1	0	$-0-P-48'FFFFFFFFFFFFFF-0-1$
0	0	0	1	0	1	1	$-0-A:B-48'FFFFFFFFFFFFFF-0-1$
0	0	0	1	1	0	0	$-0-0-C-0-1$
0	0	0	1	1	0	1	$-0-PP1-C-0-1$
0	0	0	1	1	1	0	$-0-P-C-0-1$
0	0	0	1	1	1	1	$-0-A:B-C-0-1$
0	0	1	0	0	0	0	$-PCIN-0-0-0-1$
0	0	1	0	0	0	1	$-PCIN-PP1-0-0-1$
0	0	1	0	0	1	0	$-PCIN-P-0-0-1$
0	0	1	0	0	1	1	$-PCIN-A:B-0-0-1$
0	0	1	0	1	0	0	$-PCIN-0-PP2-0-1$
0	0	1	0	1	0	1	$-PCIN-PP1-PP2-0-1$
0	0	1	0	1	1	0	$-PCIN-P-PP2-0-1$
0	0	1	0	1	1	1	$-PCIN-A:B-PP2-0-1$
0	0	1	1	0	0	0	$-PCIN-0-48'FFFFFFFFFFFFFF-0-1$
0	0	1	1	0	0	1	$-PCIN-PP1-48'FFFFFFFFFFFFFF-0-1$
0	0	1	1	0	1	0	$-PCIN-P-48'FFFFFFFFFFFFFF-0-1$
0	0	1	1	0	1	1	$-PCIN-A:B-48'FFFFFFFFFFFFFF-0-1$
0	0	1	1	1	0	0	$-PCIN-0-C-0-1$
0	0	1	1	1	0	1	$-PCIN-PP1-C-0-1$
0	0	1	1	1	1	0	$-PCIN-P-C-0-1$
0	0	1	1	1	1	1	$-PCIN-A:B-C-0-1$
0	1	0	0	0	0	0	$-P-0-0-0-1$
0	1	0	0	0	0	1	$-P-PP1-0-0-1$
0	1	0	0	0	1	0	$-P-P-0-0-1$
0	1	0	0	0	1	1	$-P-A:B-0-0-1$
0	1	0	0	1	0	0	$-P-0-PP2-0-1$
0	1	0	0	1	0	1	$-P-PP1-PP2-0-1$
0	1	0	0	1	1	0	$-P-P-PP2-0-1$
0	1	0	0	1	1	1	$-P-A:B-PP2-0-1$
0	1	0	1	0	0	0	$-P-0-48'FFFFFFFFFFFFFF-0-1$
0	1	0	1	0	0	1	$-P-PP1-48'FFFFFFFFFFFFFF-0-1$
0	1	0	1	0	1	0	$-P-P-48'FFFFFFFFFFFFFF-0-1$
0	1	0	1	0	1	1	$-P-A:B-48'FFFFFFFFFFFFFF-0-1$
0	1	0	1	1	0	0	$-P-0-C-0-1$
0	1	0	1	1	0	1	$-P-PP1-C-0-1$

Continued on next page

Table 8.7: ALUMODE 0011 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	1	0	1	1	1	0	$-P - P - C - 0 - 1$
0	1	0	1	1	1	1	$-P - A : B - C - 0 - 1$
0	1	1	0	0	0	0	$-C - 0 - 0 - 0 - 1$
0	1	1	0	0	0	1	$-C - PP1 - 0 - 0 - 1$
0	1	1	0	0	1	0	$-C - P - 0 - 0 - 1$
0	1	1	0	0	1	1	$-C - A : B - 0 - 0 - 1$
0	1	1	0	1	0	0	$-C - 0 - PP2 - 0 - 1$
0	1	1	0	1	0	1	$-C - PP1 - PP2 - 0 - 1$
0	1	1	0	1	1	0	$-C - P - PP2 - 0 - 1$
0	1	1	0	1	1	1	$-C - A : B - PP2 - 0 - 1$
0	1	1	1	0	0	0	$-C - 0 - 48'FFFFFFFFFFFFFFF - 0 - 1$
0	1	1	1	0	0	1	$-C - PP1 - 48'FFFFFFFFFFFFFFF - 0 - 1$
0	1	1	1	0	1	0	$-C - P - 48'FFFFFFFFFFFFFFF - 0 - 1$
0	1	1	1	0	1	1	$-C - A : B - 48'FFFFFFFFFFFFFFF - 0 - 1$
0	1	1	1	1	0	0	$-C - 0 - C - 0 - 1$
0	1	1	1	1	0	1	$-C - PP1 - C - 0 - 1$
0	1	1	1	1	1	0	$-C - P - C - 0 - 1$
0	1	1	1	1	1	1	$-C - A : B - C - 0 - 1$
1	0	0	0	0	0	0	$-P - 0 - 0 - 0 - 1$
1	0	0	0	0	0	1	$-P - PP1 - 0 - 0 - 1$
1	0	0	0	0	1	0	$-P - P - 0 - 0 - 1$
1	0	0	0	0	1	1	$-P - A : B - 0 - 0 - 1$
1	0	0	0	1	0	0	$-P - 0 - PP2 - 0 - 1$
1	0	0	0	1	0	1	$-P - PP1 - PP2 - 0 - 1$
1	0	0	0	1	1	0	$-P - P - PP2 - 0 - 1$
1	0	0	0	1	1	1	$-P - A : B - PP2 - 0 - 1$
1	0	0	1	0	0	0	$-P - 0 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	0	1	$-P - PP1 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	1	0	$-P - P - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	0	1	1	$-P - A : B - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	0	1	1	0	0	$-P - 0 - C - 0 - 1$
1	0	0	1	1	0	1	$-P - PP1 - C - 0 - 1$
1	0	0	1	1	1	0	$-P - P - C - 0 - 1$
1	0	0	1	1	1	1	$-P - A : B - C - 0 - 1$
1	0	1	0	0	0	0	$-RS\_PCIN - 0 - 0 - 0 - 1$
1	0	1	0	0	0	1	$-RS\_PCIN - PP1 - 0 - 0 - 1$
1	0	1	0	0	1	0	$-RS\_PCIN - P - 0 - 0 - 1$
1	0	1	0	0	1	1	$-RS\_PCIN - A : B - 0 - 0 - 1$
1	0	1	0	1	0	0	$-RS\_PCIN - 0 - PP2 - 0 - 1$
1	0	1	0	1	0	1	$-RS\_PCIN - PP1 - PP2 - 0 - 1$
1	0	1	0	1	1	0	$-RS\_PCIN - P - PP2 - 0 - 1$
1	0	1	0	1	1	1	$-RS\_PCIN - A : B - PP2 - 0 - 1$
1	0	1	1	0	0	0	$-RS\_PCIN - 0 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	0	1	$-RS\_PCIN - PP1 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	1	0	$-RS\_PCIN - P - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	0	1	1	$-RS\_PCIN - A : B - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	0	1	1	1	0	0	$-RS\_PCIN - 0 - C - 0 - 1$
1	0	1	1	1	0	1	$-RS\_PCIN - PP1 - C - 0 - 1$
1	0	1	1	1	1	0	$-RS\_PCIN - P - C - 0 - 1$
1	0	1	1	1	1	1	$-RS\_PCIN - A : B - C - 0 - 1$
1	1	0	0	0	0	0	$-RS\_P - 0 - 0 - 0 - 1$
1	1	0	0	0	0	1	$-RS\_P - PP1 - 0 - 0 - 1$
1	1	0	0	0	1	0	$-RS\_P - P - 0 - 0 - 1$
1	1	0	0	0	1	1	$-RS\_P - A : B - 0 - 0 - 1$
1	1	0	0	1	0	0	$-RS\_P - 0 - PP2 - 0 - 1$
1	1	0	0	1	0	1	$-RS\_P - PP1 - PP2 - 0 - 1$
1	1	0	0	1	1	0	$-RS\_P - P - PP2 - 0 - 1$
1	1	0	0	1	1	1	$-RS\_P - A : B - PP2 - 0 - 1$
1	1	0	1	0	0	0	$-RS\_P - 0 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	0	0	1	$-RS\_P - PP1 - 48'FFFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	0	1	0	$-RS\_P - P - 48'FFFFFFFFFFFFFFF - 0 - 1$

Continued on next page

Table 8.7: ALUMODE 0011 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	1	0	1	0	1	1	$-RS\_P - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	0	1	1	0	0	$-RS\_P - 0 - C - 0 - 1$
1	1	0	1	1	0	1	$-RS\_P - PP1 - C - 0 - 1$
1	1	0	1	1	1	0	$-RS\_P - P - C - 0 - 1$
1	1	0	1	1	1	1	$-RS\_P - A : B - C - 0 - 1$
1	1	1	0	0	0	0	$-RS\_P - 0 - 0 - 0 - 1$
1	1	1	0	0	0	1	$-RS\_P - PP1 - 0 - 0 - 1$
1	1	1	0	0	1	0	$-RS\_P - P - 0 - 0 - 1$
1	1	1	0	0	1	1	$-RS\_P - A : B - 0 - 0 - 1$
1	1	1	0	1	0	0	$-RS\_P - 0 - PP2 - 0 - 1$
1	1	1	0	1	0	1	$-RS\_P - PP1 - PP2 - 0 - 1$
1	1	1	0	1	1	0	$-RS\_P - P - PP2 - 0 - 1$
1	1	1	0	1	1	1	$-RS\_P - A : B - PP2 - 0 - 1$
1	1	1	1	0	0	0	$-RS\_P - 0 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	0	0	1	$-RS\_P - PP1 - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	0	1	0	$-RS\_P - P - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	0	1	1	$-RS\_P - A : B - 48'FFFFFFFFFFFFFF - 0 - 1$
1	1	1	1	1	0	0	$-RS\_P - 0 - C - 0 - 1$
1	1	1	1	1	0	1	$-RS\_P - PP1 - C - 0 - 1$
1	1	1	1	1	1	0	$-RS\_P - P - C - 0 - 1$
1	1	1	1	1	1	1	$-RS\_P - A : B - C - 0 - 1$

Table 8.8: ALUMODE 0100 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$0 \oplus 0 \oplus 0$
0	0	0	0	0	0	1	$0 \oplus 0 \oplus PP1$
0	0	0	0	0	1	0	$0 \oplus 0 \oplus P$
0	0	0	0	0	1	1	$0 \oplus 0 \oplus A : B$
0	0	0	0	1	0	0	$0 \oplus PP2 \oplus 0$
0	0	0	0	1	0	1	$0 \oplus PP2 \oplus PP1$
0	0	0	0	1	1	0	$0 \oplus PP2 \oplus P$
0	0	0	0	1	1	1	$0 \oplus PP2 \oplus A : B$
0	0	0	1	0	0	0	$0 \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	0	0	1	0	0	1	$0 \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	0	0	1	0	1	0	$0 \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	0	0	1	0	1	1	$0 \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	0	0	1	1	0	0	$0 \oplus C \oplus 0$
0	0	0	1	1	0	1	$0 \oplus C \oplus PP1$
0	0	0	1	1	1	0	$0 \oplus C \oplus P$
0	0	0	1	1	1	1	$0 \oplus C \oplus A : B$
0	0	1	0	0	0	0	$PCIN \oplus 0 \oplus 0$
0	0	1	0	0	0	1	$PCIN \oplus 0 \oplus PP1$
0	0	1	0	0	1	0	$PCIN \oplus 0 \oplus P$
0	0	1	0	0	1	1	$PCIN \oplus 0 \oplus A : B$
0	0	1	0	1	0	0	$PCIN \oplus PP2 \oplus 0$
0	0	1	0	1	0	1	$PCIN \oplus PP2 \oplus PP1$
0	0	1	0	1	1	0	$PCIN \oplus PP2 \oplus P$
0	0	1	0	1	1	1	$PCIN \oplus PP2 \oplus A : B$
0	0	1	1	0	0	0	$PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	0	1	1	0	0	1	$PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	0	1	1	0	1	0	$PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	0	1	1	0	1	1	$PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	0	1	1	1	0	0	$PCIN \oplus C \oplus 0$
0	0	1	1	1	0	1	$PCIN \oplus C \oplus PP1$
0	0	1	1	1	1	0	$PCIN \oplus C \oplus P$

Continued on next page

Table 8.8: ALUMODE 0100 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	0	1	1	1	1		$PCIN \oplus C \oplus A : B$
0	1	0	0	0	0	0	$P \oplus 0 \oplus 0$
0	1	0	0	0	0	1	$P \oplus 0 \oplus PP1$
0	1	0	0	0	1	0	$P \oplus 0 \oplus P$
0	1	0	0	0	1	1	$P \oplus 0 \oplus A : B$
0	1	0	0	1	0	0	$P \oplus PP2 \oplus 0$
0	1	0	0	1	0	1	$P \oplus PP2 \oplus PP1$
0	1	0	0	1	1	0	$P \oplus PP2 \oplus P$
0	1	0	0	1	1	1	$P \oplus PP2 \oplus A : B$
0	1	0	1	0	0	0	$P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	1	0	1	0	0	1	$P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	1	0	1	0	1	0	$P \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	1	0	1	0	1	1	$P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	1	0	1	1	0	0	$P \oplus C \oplus 0$
0	1	0	1	1	0	1	$P \oplus C \oplus PP1$
0	1	0	1	1	1	0	$P \oplus C \oplus P$
0	1	0	1	1	1	1	$P \oplus C \oplus A : B$
0	1	1	0	0	0	0	$C \oplus 0 \oplus 0$
0	1	1	0	0	0	1	$C \oplus 0 \oplus PP1$
0	1	1	0	0	1	0	$C \oplus 0 \oplus P$
0	1	1	0	0	1	1	$C \oplus 0 \oplus A : B$
0	1	1	0	1	0	0	$C \oplus PP2 \oplus 0$
0	1	1	0	1	0	1	$C \oplus PP2 \oplus PP1$
0	1	1	0	1	1	0	$C \oplus PP2 \oplus P$
0	1	1	0	1	1	1	$C \oplus PP2 \oplus A : B$
0	1	1	1	0	0	0	$C \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	1	1	1	0	0	1	$C \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	1	1	1	0	1	0	$C \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	1	1	1	0	1	1	$C \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	1	1	1	1	0	0	$C \oplus C \oplus 0$
0	1	1	1	1	0	1	$C \oplus C \oplus PP1$
0	1	1	1	1	1	0	$C \oplus C \oplus P$
0	1	1	1	1	1	1	$C \oplus C \oplus A : B$
1	0	0	0	0	0	0	$P \oplus 0 \oplus 0$
1	0	0	0	0	0	1	$P \oplus 0 \oplus PP1$
1	0	0	0	0	1	0	$P \oplus 0 \oplus P$
1	0	0	0	0	1	1	$P \oplus 0 \oplus A : B$
1	0	0	0	1	0	0	$P \oplus PP2 \oplus 0$
1	0	0	0	1	0	1	$P \oplus PP2 \oplus PP1$
1	0	0	0	1	1	0	$P \oplus PP2 \oplus P$
1	0	0	0	1	1	1	$P \oplus PP2 \oplus A : B$
1	0	0	1	0	0	0	$P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	0	0	1	0	0	1	$P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	0	0	1	0	1	0	$P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	0	0	1	0	1	1	$P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	0	0	1	1	0	0	$P \oplus C \oplus 0$
1	0	0	1	1	0	1	$P \oplus C \oplus PP1$
1	0	0	1	1	1	0	$P \oplus C \oplus P$
1	0	0	1	1	1	1	$P \oplus C \oplus A : B$
1	0	1	0	0	0	0	$RS\_PCIN \oplus 0 \oplus 0$
1	0	1	0	0	0	1	$RS\_PCIN \oplus 0 \oplus PP1$
1	0	1	0	0	1	0	$RS\_PCIN \oplus 0 \oplus P$
1	0	1	0	0	1	1	$RS\_PCIN \oplus 0 \oplus A : B$
1	0	1	0	1	0	0	$RS\_PCIN \oplus PP2 \oplus 0$
1	0	1	0	1	0	1	$RS\_PCIN \oplus PP2 \oplus PP1$
1	0	1	0	1	1	0	$RS\_PCIN \oplus PP2 \oplus P$
1	0	1	0	1	1	1	$RS\_PCIN \oplus PP2 \oplus A : B$
1	0	1	1	0	0	0	$RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	0	1	1	0	0	1	$RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	0	1	1	0	1	0	$RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	0	1	1	0	1	1	$RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B$

Continued on next page

Table 8.8: ALUMODE 0100 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	0	1	1	1	0	0	$RS\_PCIN \oplus C \oplus 0$
1	0	1	1	1	0	1	$RS\_PCIN \oplus C \oplus PP1$
1	0	1	1	1	1	0	$RS\_PCIN \oplus C \oplus P$
1	0	1	1	1	1	1	$RS\_PCIN \oplus C \oplus A : B$
1	1	0	0	0	0	0	$RS\_P \oplus 0 \oplus 0$
1	1	0	0	0	0	1	$RS\_P \oplus 0 \oplus PP1$
1	1	0	0	0	1	0	$RS\_P \oplus 0 \oplus P$
1	1	0	0	0	1	1	$RS\_P \oplus 0 \oplus A : B$
1	1	0	0	1	0	0	$RS\_P \oplus PP2 \oplus 0$
1	1	0	0	1	0	1	$RS\_P \oplus PP2 \oplus PP1$
1	1	0	0	1	1	0	$RS\_P \oplus PP2 \oplus P$
1	1	0	0	1	1	1	$RS\_P \oplus PP2 \oplus A : B$
1	1	0	1	0	0	0	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	1	0	1	0	0	1	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	1	0	1	0	1	0	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	1	0	1	0	1	1	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	1	0	1	1	0	0	$RS\_P \oplus C \oplus 0$
1	1	0	1	1	0	1	$RS\_P \oplus C \oplus PP1$
1	1	0	1	1	1	0	$RS\_P \oplus C \oplus P$
1	1	0	1	1	1	1	$RS\_P \oplus C \oplus A : B$
1	1	1	0	0	0	0	$RS\_P \oplus 0 \oplus 0$
1	1	1	0	0	0	1	$RS\_P \oplus 0 \oplus PP1$
1	1	1	0	0	1	0	$RS\_P \oplus 0 \oplus P$
1	1	1	0	0	1	1	$RS\_P \oplus 0 \oplus A : B$
1	1	1	0	1	0	0	$RS\_P \oplus PP2 \oplus 0$
1	1	1	0	1	0	1	$RS\_P \oplus PP2 \oplus PP1$
1	1	1	0	1	1	0	$RS\_P \oplus PP2 \oplus P$
1	1	1	0	1	1	1	$RS\_P \oplus PP2 \oplus A : B$
1	1	1	1	0	0	0	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	1	1	1	0	0	1	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	1	1	1	0	1	0	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	1	1	1	0	1	1	$RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	1	1	1	1	0	0	$RS\_P \oplus C \oplus 0$
1	1	1	1	1	0	1	$RS\_P \oplus C \oplus PP1$
1	1	1	1	1	1	0	$RS\_P \oplus C \oplus P$
1	1	1	1	1	1	1	$RS\_P \oplus C \oplus A : B$

Table 8.9: ALUMODE 0101 Observed Results

OP Modes							Observed Outputs
Z		Y		X			
0	0	0	0	0	0	0	$\neg 0 \oplus 0 \oplus 0$
0	0	0	0	0	0	1	$\neg 0 \oplus 0 \oplus PP1$
0	0	0	0	0	1	0	$\neg 0 \oplus 0 \oplus P$
0	0	0	0	0	1	1	$\neg 0 \oplus 0 \oplus A : B$
0	0	0	0	1	0	0	$\neg 0 \oplus PP2 \oplus 0$
0	0	0	0	1	0	1	$\neg 0 \oplus PP2 \oplus PP1$
0	0	0	0	1	1	0	$\neg 0 \oplus PP2 \oplus P$
0	0	0	0	1	1	1	$\neg 0 \oplus PP2 \oplus A : B$
0	0	0	1	0	0	0	$\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	0	0	1	0	0	1	$\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	0	0	1	0	1	0	$\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	0	0	1	0	1	1	$\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	0	0	1	1	0	0	$\neg 0 \oplus C \oplus 0$
0	0	0	1	1	0	1	$\neg 0 \oplus C \oplus PP1$
0	0	0	1	1	1	0	$\neg 0 \oplus C \oplus P$
0	0	0	1	1	1	1	$\neg 0 \oplus C \oplus A : B$

Continued on next page

Table 8.9: ALUMODE 0101 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	0	1	0	0	0	0	$\neg PCIN \oplus 0 \oplus 0$
0	0	1	0	0	0	1	$\neg PCIN \oplus 0 \oplus PP1$
0	0	1	0	0	1	0	$\neg PCIN \oplus 0 \oplus P$
0	0	1	0	0	1	1	$\neg PCIN \oplus 0 \oplus A : B$
0	0	1	0	1	0	0	$\neg PCIN \oplus PP2 \oplus 0$
0	0	1	0	1	0	1	$\neg PCIN \oplus PP2 \oplus PP1$
0	0	1	0	1	1	0	$\neg PCIN \oplus PP2 \oplus P$
0	0	1	0	1	1	1	$\neg PCIN \oplus PP2 \oplus A : B$
0	0	1	1	0	0	0	$\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	0	1	1	0	0	1	$\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	0	1	1	0	1	0	$\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	0	1	1	0	1	1	$\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	0	1	1	1	0	0	$\neg PCIN \oplus C \oplus 0$
0	0	1	1	1	0	1	$\neg PCIN \oplus C \oplus PP1$
0	0	1	1	1	1	0	$\neg PCIN \oplus C \oplus P$
0	0	1	1	1	1	1	$\neg PCIN \oplus C \oplus A : B$
0	1	0	0	0	0	0	$\neg P \oplus 0 \oplus 0$
0	1	0	0	0	0	1	$\neg P \oplus 0 \oplus PP1$
0	1	0	0	0	1	0	$\neg P \oplus 0 \oplus P$
0	1	0	0	0	1	1	$\neg P \oplus 0 \oplus A : B$
0	1	0	0	1	0	0	$\neg P \oplus PP2 \oplus 0$
0	1	0	0	1	0	1	$\neg P \oplus PP2 \oplus PP1$
0	1	0	0	1	1	0	$\neg P \oplus PP2 \oplus P$
0	1	0	0	1	1	1	$\neg P \oplus PP2 \oplus A : B$
0	1	0	1	0	0	0	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	1	0	1	0	0	1	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	1	0	1	0	1	0	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	1	0	1	0	1	1	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	1	0	1	1	0	0	$\neg P \oplus C \oplus 0$
0	1	0	1	1	0	1	$\neg P \oplus C \oplus PP1$
0	1	0	1	1	1	0	$\neg P \oplus C \oplus P$
0	1	0	1	1	1	1	$\neg P \oplus C \oplus A : B$
0	1	1	0	0	0	0	$\neg C \oplus 0 \oplus 0$
0	1	1	0	0	0	1	$\neg C \oplus 0 \oplus PP1$
0	1	1	0	0	1	0	$\neg C \oplus 0 \oplus P$
0	1	1	0	0	1	1	$\neg C \oplus 0 \oplus A : B$
0	1	1	0	1	0	0	$\neg C \oplus PP2 \oplus 0$
0	1	1	0	1	0	1	$\neg C \oplus PP2 \oplus PP1$
0	1	1	0	1	1	0	$\neg C \oplus PP2 \oplus P$
0	1	1	0	1	1	1	$\neg C \oplus PP2 \oplus A : B$
0	1	1	1	0	0	0	$\neg C \oplus 48'FFFFFFFFFFFFFF \oplus 0$
0	1	1	1	0	0	1	$\neg C \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
0	1	1	1	0	1	0	$\neg C \oplus 48'FFFFFFFFFFFFFF \oplus P$
0	1	1	1	0	1	1	$\neg C \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
0	1	1	1	1	0	0	$\neg C \oplus C \oplus 0$
0	1	1	1	1	0	1	$\neg C \oplus C \oplus PP1$
0	1	1	1	1	1	0	$\neg C \oplus C \oplus P$
0	1	1	1	1	1	1	$\neg C \oplus C \oplus A : B$
1	0	0	0	0	0	0	$\neg P \oplus 0 \oplus 0$
1	0	0	0	0	0	1	$\neg P \oplus 0 \oplus PP1$
1	0	0	0	0	1	0	$\neg P \oplus 0 \oplus P$
1	0	0	0	0	1	1	$\neg P \oplus 0 \oplus A : B$
1	0	0	0	1	0	0	$\neg P \oplus PP2 \oplus 0$
1	0	0	0	1	0	1	$\neg P \oplus PP2 \oplus PP1$
1	0	0	0	1	1	0	$\neg P \oplus PP2 \oplus P$
1	0	0	0	1	1	1	$\neg P \oplus PP2 \oplus A : B$
1	0	0	1	0	0	0	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	0	0	1	0	0	1	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	0	0	1	0	1	0	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	0	0	1	0	1	1	$\neg P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	0	0	1	1	0	0	$\neg P \oplus C \oplus 0$

Continued on next page

Table 8.9: ALUMODE 0101 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y		X				
1	0	0	1	1	0	1	$\neg P \oplus C \oplus PP1$
1	0	0	1	1	1	0	$\neg P \oplus C \oplus P$
1	0	0	1	1	1	1	$\neg P \oplus C \oplus A : B$
1	0	1	0	0	0	0	$\neg RS\_PCIN \oplus 0 \oplus 0$
1	0	1	0	0	0	1	$\neg RS\_PCIN \oplus 0 \oplus PP1$
1	0	1	0	0	1	0	$\neg RS\_PCIN \oplus 0 \oplus P$
1	0	1	0	0	1	1	$\neg RS\_PCIN \oplus 0 \oplus A : B$
1	0	1	0	1	0	0	$\neg RS\_PCIN \oplus PP2 \oplus 0$
1	0	1	0	1	0	1	$\neg RS\_PCIN \oplus PP2 \oplus PP1$
1	0	1	0	1	1	0	$\neg RS\_PCIN \oplus PP2 \oplus P$
1	0	1	0	1	1	1	$\neg RS\_PCIN \oplus PP2 \oplus A : B$
1	0	1	1	0	0	0	$\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	0	1	1	0	0	1	$\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	0	1	1	0	1	0	$\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	0	1	1	0	1	1	$\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	0	1	1	1	0	0	$\neg RS\_PCIN \oplus C \oplus 0$
1	0	1	1	1	0	1	$\neg RS\_PCIN \oplus C \oplus PP1$
1	0	1	1	1	1	0	$\neg RS\_PCIN \oplus C \oplus P$
1	0	1	1	1	1	1	$\neg RS\_PCIN \oplus C \oplus A : B$
1	1	0	0	0	0	0	$\neg RS\_P \oplus 0 \oplus 0$
1	1	0	0	0	0	1	$\neg RS\_P \oplus 0 \oplus PP1$
1	1	0	0	0	1	0	$\neg RS\_P \oplus 0 \oplus P$
1	1	0	0	0	1	1	$\neg RS\_P \oplus 0 \oplus A : B$
1	1	0	0	1	0	0	$\neg RS\_P \oplus PP2 \oplus 0$
1	1	0	0	1	0	1	$\neg RS\_P \oplus PP2 \oplus PP1$
1	1	0	0	1	1	0	$\neg RS\_P \oplus PP2 \oplus P$
1	1	0	0	1	1	1	$\neg RS\_P \oplus PP2 \oplus A : B$
1	1	0	1	0	0	0	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	1	0	1	0	0	1	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	1	0	1	0	1	0	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	1	0	1	0	1	1	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	1	0	1	1	0	0	$\neg RS\_P \oplus C \oplus 0$
1	1	0	1	1	0	1	$\neg RS\_P \oplus C \oplus PP1$
1	1	0	1	1	1	0	$\neg RS\_P \oplus C \oplus P$
1	1	0	1	1	1	1	$\neg RS\_P \oplus C \oplus A : B$
1	1	1	0	0	0	0	$\neg RS\_P \oplus 0 \oplus 0$
1	1	1	0	0	0	1	$\neg RS\_P \oplus 0 \oplus PP1$
1	1	1	0	0	1	0	$\neg RS\_P \oplus 0 \oplus P$
1	1	1	0	0	1	1	$\neg RS\_P \oplus 0 \oplus A : B$
1	1	1	0	1	0	0	$\neg RS\_P \oplus PP2 \oplus 0$
1	1	1	0	1	0	1	$\neg RS\_P \oplus PP2 \oplus PP1$
1	1	1	0	1	1	0	$\neg RS\_P \oplus PP2 \oplus P$
1	1	1	0	1	1	1	$\neg RS\_P \oplus PP2 \oplus A : B$
1	1	1	1	0	0	0	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0$
1	1	1	1	0	0	1	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1$
1	1	1	1	0	1	0	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P$
1	1	1	1	0	1	1	$\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B$
1	1	1	1	1	0	0	$\neg RS\_P \oplus C \oplus 0$
1	1	1	1	1	0	1	$\neg RS\_P \oplus C \oplus PP1$
1	1	1	1	1	1	0	$\neg RS\_P \oplus C \oplus P$
1	1	1	1	1	1	1	$\neg RS\_P \oplus C \oplus A : B$

Table 8.10: ALUMODE 0110 Observed Results

OP Modes							Observed Outputs
Z		Y			X		
0	0	0	0	0	0	0	$\neg(0 \oplus 0 \oplus 0)$

Continued on next page



Table 8.10: ALUMODE 0110 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	1	$\neg(0 \oplus 0 \oplus PP1)$
0	0	0	0	0	0	1	$\neg(0 \oplus 0 \oplus P)$
0	0	0	0	0	0	1	$\neg(0 \oplus 0 \oplus A : B)$
0	0	0	0	0	1	0	$\neg(0 \oplus PP2 \oplus 0)$
0	0	0	0	0	1	0	$\neg(0 \oplus PP2 \oplus PP1)$
0	0	0	0	0	1	0	$\neg(0 \oplus PP2 \oplus P)$
0	0	0	0	0	1	1	$\neg(0 \oplus PP2 \oplus A : B)$
0	0	0	0	1	0	0	$\neg(0 \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	0	0	1	0	0	$\neg(0 \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	0	0	1	0	1	$\neg(0 \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	0	0	1	0	1	$\neg(0 \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	0	0	1	1	0	$\neg(0 \oplus C \oplus 0)$
0	0	0	0	1	1	0	$\neg(0 \oplus C \oplus PP1)$
0	0	0	0	1	1	0	$\neg(0 \oplus C \oplus P)$
0	0	0	0	1	1	1	$\neg(0 \oplus C \oplus A : B)$
0	0	0	1	0	0	0	$\neg(PCIN \oplus 0 \oplus 0)$
0	0	0	1	0	0	0	$\neg(PCIN \oplus 0 \oplus PP1)$
0	0	0	1	0	0	1	$\neg(PCIN \oplus 0 \oplus P)$
0	0	0	1	0	0	1	$\neg(PCIN \oplus 0 \oplus A : B)$
0	0	0	1	0	1	0	$\neg(PCIN \oplus PP2 \oplus 0)$
0	0	0	1	0	1	0	$\neg(PCIN \oplus PP2 \oplus PP1)$
0	0	0	1	0	1	0	$\neg(PCIN \oplus PP2 \oplus P)$
0	0	0	1	0	1	1	$\neg(PCIN \oplus PP2 \oplus A : B)$
0	0	0	1	1	0	0	$\neg(PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	0	1	1	0	0	$\neg(PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	0	1	1	0	1	$\neg(PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	0	1	1	0	1	$\neg(PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	0	1	1	1	0	$\neg(PCIN \oplus C \oplus 0)$
0	0	0	1	1	1	0	$\neg(PCIN \oplus C \oplus PP1)$
0	0	0	1	1	1	0	$\neg(PCIN \oplus C \oplus P)$
0	0	0	1	1	1	1	$\neg(PCIN \oplus C \oplus A : B)$
0	0	1	0	0	0	0	$\neg(P \oplus 0 \oplus 0)$
0	0	1	0	0	0	0	$\neg(P \oplus 0 \oplus PP1)$
0	0	1	0	0	0	1	$\neg(P \oplus 0 \oplus P)$
0	0	1	0	0	0	1	$\neg(P \oplus 0 \oplus A : B)$
0	0	1	0	0	1	0	$\neg(P \oplus PP2 \oplus 0)$
0	0	1	0	0	1	0	$\neg(P \oplus PP2 \oplus PP1)$
0	0	1	0	0	1	0	$\neg(P \oplus PP2 \oplus P)$
0	0	1	0	0	1	1	$\neg(P \oplus PP2 \oplus A : B)$
0	0	1	0	1	0	0	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	1	0	1	0	0	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	1	0	1	0	1	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	1	0	1	0	1	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	1	0	1	1	0	$\neg(P \oplus C \oplus 0)$
0	0	1	0	1	1	0	$\neg(P \oplus C \oplus PP1)$
0	0	1	0	1	1	0	$\neg(P \oplus C \oplus P)$
0	0	1	0	1	1	1	$\neg(P \oplus C \oplus A : B)$
0	0	1	1	0	0	0	$\neg(C \oplus 0 \oplus 0)$
0	0	1	1	0	0	0	$\neg(C \oplus 0 \oplus PP1)$
0	0	1	1	0	0	1	$\neg(C \oplus 0 \oplus P)$
0	0	1	1	0	0	1	$\neg(C \oplus 0 \oplus A : B)$
0	0	1	1	0	1	0	$\neg(C \oplus PP2 \oplus 0)$
0	0	1	1	0	1	0	$\neg(C \oplus PP2 \oplus PP1)$
0	0	1	1	0	1	0	$\neg(C \oplus PP2 \oplus P)$
0	0	1	1	0	1	1	$\neg(C \oplus PP2 \oplus A : B)$
0	0	1	1	1	0	0	$\neg(C \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	1	1	1	0	0	$\neg(C \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	1	1	1	0	1	$\neg(C \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	1	1	1	0	1	$\neg(C \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	1	1	1	1	0	$\neg(C \oplus C \oplus 0)$
0	0	1	1	1	1	0	$\neg(C \oplus C \oplus PP1)$

Continued on next page

Table 8.10: ALUMODE 0110 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z	Y			X			
0	1	1	1	1	1	0	$\neg(C \oplus C \oplus P)$
0	1	1	1	1	1	1	$\neg(C \oplus C \oplus A : B)$
1	0	0	0	0	0	0	$\neg(P \oplus 0 \oplus 0)$
1	0	0	0	0	0	1	$\neg(P \oplus 0 \oplus PP1)$
1	0	0	0	0	1	0	$\neg(P \oplus 0 \oplus P)$
1	0	0	0	0	1	1	$\neg(P \oplus 0 \oplus A : B)$
1	0	0	0	1	0	0	$\neg(P \oplus PP2 \oplus 0)$
1	0	0	0	1	0	1	$\neg(P \oplus PP2 \oplus PP1)$
1	0	0	0	1	1	0	$\neg(P \oplus PP2 \oplus P)$
1	0	0	0	1	1	1	$\neg(P \oplus PP2 \oplus A : B)$
1	0	0	1	0	0	0	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	0	0	1	0	0	1	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	0	0	1	0	1	0	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	0	0	1	0	1	1	$\neg(P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	0	0	1	1	0	0	$\neg(P \oplus C \oplus 0)$
1	0	0	1	1	0	1	$\neg(P \oplus C \oplus PP1)$
1	0	0	1	1	1	0	$\neg(P \oplus C \oplus P)$
1	0	0	1	1	1	1	$\neg(P \oplus C \oplus A : B)$
1	0	1	0	0	0	0	$\neg(RS\_PCIN \oplus 0 \oplus 0)$
1	0	1	0	0	0	1	$\neg(RS\_PCIN \oplus 0 \oplus PP1)$
1	0	1	0	0	1	0	$\neg(RS\_PCIN \oplus 0 \oplus P)$
1	0	1	0	0	1	1	$\neg(RS\_PCIN \oplus 0 \oplus A : B)$
1	0	1	0	1	0	0	$\neg(RS\_PCIN \oplus PP2 \oplus 0)$
1	0	1	0	1	0	1	$\neg(RS\_PCIN \oplus PP2 \oplus PP1)$
1	0	1	0	1	1	0	$\neg(RS\_PCIN \oplus PP2 \oplus P)$
1	0	1	0	1	1	1	$\neg(RS\_PCIN \oplus PP2 \oplus A : B)$
1	0	1	1	0	0	0	$\neg(RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	0	1	1	0	0	1	$\neg(RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	0	1	1	0	1	0	$\neg(RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	0	1	1	0	1	1	$\neg(RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	0	1	1	1	0	0	$\neg(RS\_PCIN \oplus C \oplus 0)$
1	0	1	1	1	0	1	$\neg(RS\_PCIN \oplus C \oplus PP1)$
1	0	1	1	1	1	0	$\neg(RS\_PCIN \oplus C \oplus P)$
1	0	1	1	1	1	1	$\neg(RS\_PCIN \oplus C \oplus A : B)$
1	1	0	0	0	0	0	$\neg(RS\_P \oplus 0 \oplus 0)$
1	1	0	0	0	0	1	$\neg(RS\_P \oplus 0 \oplus PP1)$
1	1	0	0	0	1	0	$\neg(RS\_P \oplus 0 \oplus P)$
1	1	0	0	0	1	1	$\neg(RS\_P \oplus 0 \oplus A : B)$
1	1	0	0	1	0	0	$\neg(RS\_P \oplus PP2 \oplus 0)$
1	1	0	0	1	0	1	$\neg(RS\_P \oplus PP2 \oplus PP1)$
1	1	0	0	1	1	0	$\neg(RS\_P \oplus PP2 \oplus P)$
1	1	0	0	1	1	1	$\neg(RS\_P \oplus PP2 \oplus A : B)$
1	1	0	1	0	0	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	1	0	1	0	0	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	1	0	1	0	1	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	1	0	1	0	1	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	1	0	1	1	0	0	$\neg(RS\_P \oplus C \oplus 0)$
1	1	0	1	1	0	1	$\neg(RS\_P \oplus C \oplus PP1)$
1	1	0	1	1	1	0	$\neg(RS\_P \oplus C \oplus P)$
1	1	0	1	1	1	1	$\neg(RS\_P \oplus C \oplus A : B)$
1	1	1	0	0	0	0	$\neg(RS\_P \oplus 0 \oplus 0)$
1	1	1	0	0	0	1	$\neg(RS\_P \oplus 0 \oplus PP1)$
1	1	1	0	0	1	0	$\neg(RS\_P \oplus 0 \oplus P)$
1	1	1	0	0	1	1	$\neg(RS\_P \oplus 0 \oplus A : B)$
1	1	1	0	1	0	0	$\neg(RS\_P \oplus PP2 \oplus 0)$
1	1	1	0	1	0	1	$\neg(RS\_P \oplus PP2 \oplus PP1)$
1	1	1	0	1	1	0	$\neg(RS\_P \oplus PP2 \oplus P)$
1	1	1	0	1	1	1	$\neg(RS\_P \oplus PP2 \oplus A : B)$
1	1	1	1	0	0	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	1	1	1	0	0	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	1	1	1	0	1	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	1	1	1	0	1	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	1	1	1	1	0	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	1	1	1	1	0	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	1	1	1	1	1	0	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	1	1	1	1	1	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$

Continued on next page

Table 8.10: ALUMODE 0110 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z		Y			X		
1	1	1	1	0	1	1	$\neg(RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	1	1	1	1	0	0	$\neg(RS\_P \oplus C \oplus 0)$
1	1	1	1	1	0	1	$\neg(RS\_P \oplus C \oplus PP1)$
1	1	1	1	1	1	0	$\neg(RS\_P \oplus C \oplus P)$
1	1	1	1	1	1	1	$\neg(RS\_P \oplus C \oplus A : B)$

Table 8.11: ALUMODE 0111 Observed Results

OP Modes							Observed Outputs	
Z	Y			X				
0	0	0	0	0	0	0	0	$\neg(\neg 0 \oplus 0 \oplus 0)$
0	0	0	0	0	0	0	1	$\neg(\neg 0 \oplus 0 \oplus PP1)$
0	0	0	0	0	0	1	0	$\neg(\neg 0 \oplus 0 \oplus P)$
0	0	0	0	0	0	1	1	$\neg(\neg 0 \oplus 0 \oplus A : B)$
0	0	0	0	0	1	0	0	$\neg(\neg 0 \oplus PP2 \oplus 0)$
0	0	0	0	0	1	0	1	$\neg(\neg 0 \oplus PP2 \oplus PP1)$
0	0	0	0	0	1	1	0	$\neg(\neg 0 \oplus PP2 \oplus P)$
0	0	0	0	0	1	1	1	$\neg(\neg 0 \oplus PP2 \oplus A : B)$
0	0	0	0	1	0	0	0	$\neg(\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	0	0	1	0	0	1	$\neg(\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	0	0	1	0	1	0	$\neg(\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	0	0	1	0	1	1	$\neg(\neg 0 \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	0	0	1	1	0	0	$\neg(\neg 0 \oplus C \oplus 0)$
0	0	0	0	1	1	0	1	$\neg(\neg 0 \oplus C \oplus PP1)$
0	0	0	0	1	1	1	0	$\neg(\neg 0 \oplus C \oplus P)$
0	0	0	0	1	1	1	1	$\neg(\neg 0 \oplus C \oplus A : B)$
0	0	0	1	0	0	0	0	$\neg(\neg PCIN \oplus 0 \oplus 0)$
0	0	0	1	0	0	0	1	$\neg(\neg PCIN \oplus 0 \oplus PP1)$
0	0	0	1	0	0	1	0	$\neg(\neg PCIN \oplus 0 \oplus P)$
0	0	0	1	0	0	1	1	$\neg(\neg PCIN \oplus 0 \oplus A : B)$
0	0	0	1	0	1	0	0	$\neg(\neg PCIN \oplus PP2 \oplus 0)$
0	0	0	1	0	1	0	1	$\neg(\neg PCIN \oplus PP2 \oplus PP1)$
0	0	0	1	0	1	1	0	$\neg(\neg PCIN \oplus PP2 \oplus P)$
0	0	0	1	0	1	1	1	$\neg(\neg PCIN \oplus PP2 \oplus A : B)$
0	0	0	1	1	0	0	0	$\neg(\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	0	0	1	1	0	0	1	$\neg(\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	0	0	1	1	0	1	0	$\neg(\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	0	0	1	1	0	1	1	$\neg(\neg PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	0	0	1	1	1	0	0	$\neg(\neg PCIN \oplus C \oplus 0)$
0	0	0	1	1	1	0	1	$\neg(\neg PCIN \oplus C \oplus PP1)$
0	0	0	1	1	1	1	0	$\neg(\neg PCIN \oplus C \oplus P)$
0	0	0	1	1	1	1	1	$\neg(\neg PCIN \oplus C \oplus A : B)$
0	1	0	0	0	0	0	0	$\neg(\neg P \oplus 0 \oplus 0)$
0	1	0	0	0	0	0	1	$\neg(\neg P \oplus 0 \oplus PP1)$
0	1	0	0	0	0	1	0	$\neg(\neg P \oplus 0 \oplus P)$
0	1	0	0	0	0	1	1	$\neg(\neg P \oplus 0 \oplus A : B)$
0	1	0	0	0	1	0	0	$\neg(\neg P \oplus PP2 \oplus 0)$
0	1	0	0	0	1	0	1	$\neg(\neg P \oplus PP2 \oplus PP1)$
0	1	0	0	0	1	1	0	$\neg(\neg P \oplus PP2 \oplus P)$
0	1	0	0	0	1	1	1	$\neg(\neg P \oplus PP2 \oplus A : B)$
0	1	0	1	0	0	0	0	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	1	0	1	0	0	0	1	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	1	0	1	0	0	1	0	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	1	0	1	0	0	1	1	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	1	0	1	1	0	0	0	$\neg(\neg P \oplus C \oplus 0)$
0	1	0	1	1	0	0	1	$\neg(\neg P \oplus C \oplus PP1)$
0	1	0	1	1	0	1	0	$\neg(\neg P \oplus C \oplus P)$

Continued on next page

Table 8.11: ALUMODE 0111 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z		Y		X			
0	1	0	1	1	1	1	$\neg(\neg P \oplus C \oplus A : B)$
0	1	1	0	0	0	0	$\neg(\neg C \oplus 0 \oplus 0)$
0	1	1	0	0	0	1	$\neg(\neg C \oplus 0 \oplus PP1)$
0	1	1	0	0	1	0	$\neg(\neg C \oplus 0 \oplus P)$
0	1	1	0	0	1	1	$\neg(\neg C \oplus 0 \oplus A : B)$
0	1	1	0	1	0	0	$\neg(\neg C \oplus PP2 \oplus 0)$
0	1	1	0	1	0	1	$\neg(\neg C \oplus PP2 \oplus PP1)$
0	1	1	0	1	1	0	$\neg(\neg C \oplus PP2 \oplus P)$
0	1	1	0	1	1	1	$\neg(\neg C \oplus PP2 \oplus A : B)$
0	1	1	1	0	0	0	$\neg(\neg C \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
0	1	1	1	0	0	1	$\neg(\neg C \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
0	1	1	1	0	1	0	$\neg(\neg C \oplus 48'FFFFFFFFFFFFFF \oplus P)$
0	1	1	1	0	1	1	$\neg(\neg C \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
0	1	1	1	1	0	0	$\neg(\neg C \oplus C \oplus 0)$
0	1	1	1	1	0	1	$\neg(\neg C \oplus C \oplus PP1)$
0	1	1	1	1	1	0	$\neg(\neg C \oplus C \oplus P)$
0	1	1	1	1	1	1	$\neg(\neg C \oplus C \oplus A : B)$
1	0	0	0	0	0	0	$\neg(\neg P \oplus 0 \oplus 0)$
1	0	0	0	0	0	1	$\neg(\neg P \oplus 0 \oplus PP1)$
1	0	0	0	0	1	0	$\neg(\neg P \oplus 0 \oplus P)$
1	0	0	0	0	1	1	$\neg(\neg P \oplus 0 \oplus A : B)$
1	0	0	0	1	0	0	$\neg(\neg P \oplus PP2 \oplus 0)$
1	0	0	0	1	0	1	$\neg(\neg P \oplus PP2 \oplus PP1)$
1	0	0	0	1	1	0	$\neg(\neg P \oplus PP2 \oplus P)$
1	0	0	0	1	1	1	$\neg(\neg P \oplus PP2 \oplus A : B)$
1	0	0	1	0	0	0	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	0	0	1	0	0	1	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	0	0	1	0	1	0	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	0	0	1	0	1	1	$\neg(\neg P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	0	0	1	1	0	0	$\neg(\neg P \oplus C \oplus 0)$
1	0	0	1	1	0	1	$\neg(\neg P \oplus C \oplus PP1)$
1	0	0	1	1	1	0	$\neg(\neg P \oplus C \oplus P)$
1	0	0	1	1	1	1	$\neg(\neg P \oplus C \oplus A : B)$
1	0	1	0	0	0	0	$\neg(\neg RS\_PCIN \oplus 0 \oplus 0)$
1	0	1	0	0	0	1	$\neg(\neg RS\_PCIN \oplus 0 \oplus PP1)$
1	0	1	0	0	1	0	$\neg(\neg RS\_PCIN \oplus 0 \oplus P)$
1	0	1	0	0	1	1	$\neg(\neg RS\_PCIN \oplus 0 \oplus A : B)$
1	0	1	0	1	0	0	$\neg(\neg RS\_PCIN \oplus PP2 \oplus 0)$
1	0	1	0	1	0	1	$\neg(\neg RS\_PCIN \oplus PP2 \oplus PP1)$
1	0	1	0	1	1	0	$\neg(\neg RS\_PCIN \oplus PP2 \oplus P)$
1	0	1	0	1	1	1	$\neg(\neg RS\_PCIN \oplus PP2 \oplus A : B)$
1	0	1	1	0	0	0	$\neg(\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	0	1	1	0	0	1	$\neg(\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	0	1	1	0	1	0	$\neg(\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	0	1	1	0	1	1	$\neg(\neg RS\_PCIN \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$
1	0	1	1	1	0	0	$\neg(\neg RS\_PCIN \oplus C \oplus 0)$
1	0	1	1	1	0	1	$\neg(\neg RS\_PCIN \oplus C \oplus PP1)$
1	0	1	1	1	1	0	$\neg(\neg RS\_PCIN \oplus C \oplus P)$
1	0	1	1	1	1	1	$\neg(\neg RS\_PCIN \oplus C \oplus A : B)$
1	1	0	0	0	0	0	$\neg(\neg RS\_P \oplus 0 \oplus 0)$
1	1	0	0	0	0	1	$\neg(\neg RS\_P \oplus 0 \oplus PP1)$
1	1	0	0	0	1	0	$\neg(\neg RS\_P \oplus 0 \oplus P)$
1	1	0	0	0	1	1	$\neg(\neg RS\_P \oplus 0 \oplus A : B)$
1	1	0	0	1	0	0	$\neg(\neg RS\_P \oplus PP2 \oplus 0)$
1	1	0	0	1	0	1	$\neg(\neg RS\_P \oplus PP2 \oplus PP1)$
1	1	0	0	1	1	0	$\neg(\neg RS\_P \oplus PP2 \oplus P)$
1	1	0	0	1	1	1	$\neg(\neg RS\_P \oplus PP2 \oplus A : B)$
1	1	0	1	0	0	0	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$
1	1	0	1	0	0	1	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$
1	1	0	1	0	1	0	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P)$
1	1	0	1	0	1	1	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$

Continued on next page

Table 8.11: ALUMODE 0111 Observed Results (*cont.*)

OP Modes								Observed Outputs
Z		Y		X				
1	1	0	1	1	0	0	$\neg(\neg RS\_P \oplus C \oplus 0)$	
1	1	0	1	1	0	1	$\neg(\neg RS\_P \oplus C \oplus PP1)$	
1	1	0	1	1	1	0	$\neg(\neg RS\_P \oplus C \oplus P)$	
1	1	0	1	1	1	1	$\neg(\neg RS\_P \oplus C \oplus A : B)$	
1	1	1	0	0	0	0	$\neg(\neg RS\_P \oplus 0 \oplus 0)$	
1	1	1	0	0	0	1	$\neg(\neg RS\_P \oplus 0 \oplus PP1)$	
1	1	1	0	0	1	0	$\neg(\neg RS\_P \oplus 0 \oplus P)$	
1	1	1	0	0	1	1	$\neg(\neg RS\_P \oplus 0 \oplus A : B)$	
1	1	1	0	1	0	0	$\neg(\neg RS\_P \oplus PP2 \oplus 0)$	
1	1	1	0	1	0	1	$\neg(\neg RS\_P \oplus PP2 \oplus PP1)$	
1	1	1	0	1	1	0	$\neg(\neg RS\_P \oplus PP2 \oplus P)$	
1	1	1	0	1	1	1	$\neg(\neg RS\_P \oplus PP2 \oplus A : B)$	
1	1	1	1	0	0	0	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus 0)$	
1	1	1	1	0	0	1	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus PP1)$	
1	1	1	1	0	1	0	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus P)$	
1	1	1	1	0	1	1	$\neg(\neg RS\_P \oplus 48'FFFFFFFFFFFFFF \oplus A : B)$	
1	1	1	1	1	0	0	$\neg(\neg RS\_P \oplus C \oplus 0)$	
1	1	1	1	1	0	1	$\neg(\neg RS\_P \oplus C \oplus PP1)$	
1	1	1	1	1	1	0	$\neg(\neg RS\_P \oplus C \oplus P)$	
1	1	1	1	1	1	1	$\neg(\neg RS\_P \oplus C \oplus A : B)$	

Table 8.12: ALUMODE 1000 Observed Results

OP Modes								Observed Outputs
Z	Y				X			
0	0	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge 0 \vee PP2 \wedge 0)$
0	0	0	1	0	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	1	0	0	0	$3 * (0 \wedge C \vee 0 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	0	1	0	$3 * (PP1 \wedge C \vee PP1 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	0	0	$3 * (P \wedge C \vee P \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	1	0	$3 * (A : B \wedge C \vee A : B \wedge 0 \vee C \wedge 0)$
0	0	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	1	0	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	0	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	0	0	$3 * (P \wedge PP2 \vee P \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	1	0	$3 * (A : B \wedge PP2 \vee A : B \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	1	0	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	1	0	0	0	$3 * (0 \wedge C \vee 0 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	0	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	0	0	$3 * (P \wedge C \vee P \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge PCIN \vee C \wedge PCIN)$

Continued on next page

Table 8.12: ALUMODE 1000 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
0	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
0	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
0	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
0	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge P \vee C \wedge P)$
0	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
0	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge C \vee 0 \wedge C)$
0	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge C \vee PP2 \wedge C)$
0	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge C \vee C \wedge C)$
0	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge C \vee C \wedge C)$
0	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge C \vee C \wedge C)$
0	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge C \vee C \wedge C)$
1	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
1	0	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
1	0	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
1	0	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
1	0	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge P \vee C \wedge P)$
1	0	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
1	0	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$

Continued on next page

Table 8.12: ALUMODE 1000 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z	Y			X			
1	0	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$

Table 8.13: ALUMODE 1001 Expected Results

OP Modes							Expected Outputs
Z	Y			X			
0	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$3 * (P \wedge 0 \vee P \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$3 * (A : B \wedge 0 \vee A : B \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	1	0	$3 * (0 \wedge PP2 \vee 0 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$3 * (P \wedge PP2 \vee P \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge 0 \vee PP2 \wedge 0)$
0	0	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge 0 \vee C \wedge 0)$
0	0	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge PCIN \vee 0 \wedge PCIN)$

Continued on next page

Table 8.13: ALUMODE 1001 Expected Results (cont.)

OP Modes							Expected Outputs
Z	Y			X			
0	0	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge PCIN \vee C \wedge PCIN)$
0	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
0	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
0	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
0	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
0	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge P \vee C \wedge P)$
0	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
0	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge C \vee 0 \wedge C)$
0	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge C \vee PP2 \wedge C)$
0	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge C \vee C \wedge C)$
0	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge C \vee C \wedge C)$
0	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge C \vee C \wedge C)$
0	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge C \vee C \wedge C)$
1	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
1	0	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
1	0	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
1	0	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$

Continued on next page



Table 8.13: ALUMODE 1001 Expected Results (cont.)

OP Modes							Expected Outputs
Z	Y			X			
1	0	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge P \vee C \wedge P)$
1	0	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
1	0	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$

Table 8.14: ALUMODE 1010 Expected Results

OP Modes							Expected Outputs
Z			Y		X		
0	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg 0 \vee 0 \wedge \neg 0)$

Continued on next page

Table 8.14: ALUMODE 1010 Expected Results (cont.)

OP Modes						Expected Outputs	
Z	Y				X		
0	0	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
0	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg C \vee C \wedge \neg C)$

Continued on next page

Table 8.14: ALUMODE 1010 Expected Results (cont.)

OP Modes							Expected Outputs
Z	Y			X			
0	1	1	1	1	1		$3 * (A : B \wedge C \vee A : B \wedge \neg C \vee C \wedge \neg C)$
1	0	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
1	0	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	1	0	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	0	0	0	0	$3 * (0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	0	1	$3 * (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	0	$3 * (P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	1	$3 * (A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	1	0	0	$3 * (0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	0	1	$3 * (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	0	$3 * (P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	1	$3 * (A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	1	0	0	0	$3 * (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	0	1	$3 * (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	0	$3 * (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	1	$3 * (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$

Continued on next page

Table 8.14: ALUMODE 1010 Expected Results (cont.)

OP Modes							Expected Outputs
Z	Y				X		
1	1	1	1	1	0	0	$3 * (0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	0	1	$3 * (PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	0	$3 * (P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	1	$3 * (A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

Table 8.15: ALUMODE 1011 Expected Results

OP Modes							Expected Outputs
Z	Y			X			
0	0	0	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$3 * \neg(P \wedge 0 \vee P \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	1	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	0	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	1	$3 * \neg(P \wedge PP2 \vee P \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	1	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	1	0	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$

Continued on next page

Table 8.15: ALUMODE 1011 Expected Results (cont.)

OP Modes							Expected Outputs
Z	Y		X				
0	1	1	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg C \vee C \wedge \neg C)$
1	0	0	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
1	0	1	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	1	$3 * \neg(PP1 \wedge C \vee PP1 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	0	$3 * \neg(P \wedge C \vee P \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	1	$3 * \neg(A : B \wedge C \vee A : B \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	1	0	0	0	0	0	$3 * \neg(0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	0	1	$3 * \neg(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	0	$3 * \neg(P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	1	$3 * \neg(A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	1	0	0	$3 * \neg(0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	0	1	$3 * \neg(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	0	$3 * \neg(P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	1	$3 * \neg(A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	1	0	0	0	$3 * \neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	0	1	$3 * \neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	0	$3 * \neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	1	$3 * \neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	1	0	0	$3 * \neg(0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

Continued on next page

Table 8.15: ALUMODE 1011 Expected Results (cont.)

OP Modes							Expected Outputs
	Z	Y			X		
1	1	0	1	1	0	1	$3 * \neg (PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	0	$3 * \neg (P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	1	$3 * \neg (A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	0	0	0	0	$3 * \neg (0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	0	1	$3 * \neg (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	0	$3 * \neg (P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	1	$3 * \neg (A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	1	0	0	$3 * \neg (0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	0	1	$3 * \neg (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	0	$3 * \neg (P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	1	$3 * \neg (A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	1	0	0	0	$3 * \neg (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	0	1	$3 * \neg (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	0	$3 * \neg (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	1	$3 * \neg (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	1	0	0	$3 * \neg (0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	0	1	$3 * \neg (PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	0	$3 * \neg (P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	1	$3 * \neg (A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

Table 8.16: ALUMODE 1100 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$(P \wedge 0 \vee P \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$(A : B \wedge 0 \vee A : B \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	1	0	$(0 \wedge PP2 \vee 0 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	0	$(PP1 \wedge PP2 \vee PP1 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$(P \wedge PP2 \vee P \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$(A : B \wedge PP2 \vee A : B \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	1	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	0	1	0	0	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	0	1	0	1	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	0	1	0	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	0	1	1	0	$(0 \wedge C \vee 0 \wedge 0 \vee C \wedge 0)$
0	0	0	0	1	1	0	$(PP1 \wedge C \vee PP1 \wedge 0 \vee C \wedge 0)$
0	0	0	0	1	1	1	$(P \wedge C \vee P \wedge 0 \vee C \wedge 0)$
0	0	0	0	1	1	1	$(A : B \wedge C \vee A : B \wedge 0 \vee C \wedge 0)$
0	0	0	1	0	0	0	$(0 \wedge 0 \vee 0 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	0	1	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	0	1	0	0	1	$(P \wedge 0 \vee P \wedge PCIN \vee 0 \wedge PCIN)$
0	0	0	1	0	0	1	$(A : B \wedge 0 \vee A : B \wedge PCIN \vee 0 \wedge PCIN)$
0	0	0	1	0	1	0	$(0 \wedge PP2 \vee 0 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	0	1	0	1	0	$(PP1 \wedge PP2 \vee PP1 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	0	1	0	1	1	$(P \wedge PP2 \vee P \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	0	1	0	1	1	$(A : B \wedge PP2 \vee A : B \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	0	1	1	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	0	1	1	0	0	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	0	1	1	0	1	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	0	1	1	0	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	0	1	1	1	0	$(0 \wedge C \vee 0 \wedge PCIN \vee C \wedge PCIN)$
0	0	0	1	1	1	0	$(PP1 \wedge C \vee PP1 \wedge PCIN \vee C \wedge PCIN)$
0	0	0	1	1	1	1	$(P \wedge C \vee P \wedge PCIN \vee C \wedge PCIN)$
0	0	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge PCIN \vee C \wedge PCIN)$
0	1	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$

Continued on next page

Table 8.16: ALUMODE 1100 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	1	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	0	$(P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
0	1	0	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	0	$(P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
0	1	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
0	1	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
0	1	0	1	1	1	0	$(P \wedge C \vee P \wedge P \vee C \wedge P)$
0	1	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
0	1	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	0	$(P \wedge 0 \vee P \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge C \vee 0 \wedge C)$
0	1	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge C \vee PP2 \wedge C)$
0	1	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge C \vee C \wedge C)$
0	1	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge C \vee C \wedge C)$
0	1	1	1	1	1	0	$(P \wedge C \vee P \wedge C \vee C \wedge C)$
0	1	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge C \vee C \wedge C)$
1	0	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	0	$(P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
1	0	0	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	0	$(P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
1	0	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
1	0	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
1	0	0	1	1	1	0	$(P \wedge C \vee P \wedge P \vee C \wedge P)$
1	0	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
1	0	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	0	$(P \wedge 0 \vee P \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$

Continued on next page



Table 8.16: ALUMODE 1100 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	0	1	1	1	0	$(P \wedge C \vee P \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$	
1	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$	
1	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	0	0	0	1	$(P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	0	0	0	1	$(A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	0	0	1	0	$(0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	0	0	1	0	$(PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	0	0	1	1	$(P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	0	0	1	1	$(A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	0	1	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	0	1	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	0	1	0	1	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	0	1	0	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	0	1	1	0	$(0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	0	1	1	0	$(PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	0	1	1	1	$(P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	0	1	1	1	$(A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	1	0	0	0	$(0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	1	0	0	0	$(PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	1	0	0	1	$(P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	1	0	0	1	$(A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$	
1	1	1	0	1	0	$(0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	1	0	1	0	$(PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	1	0	1	1	$(P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	1	0	1	1	$(A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$	
1	1	1	1	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	1	1	0	0	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	1	1	0	1	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	1	1	0	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$	
1	1	1	1	1	0	$(0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	1	1	1	0	$(PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	1	1	1	1	$(P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$	
1	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$	

Table 8.17: ALUMODE 1101 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$(P \wedge 0 \vee P \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$(A : B \wedge 0 \vee A : B \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	1	0	$(0 \wedge PP2 \vee 0 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	0	$(PP1 \wedge PP2 \vee PP1 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	1	$(P \wedge PP2 \vee P \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	0	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	0	$(P \wedge C \vee P \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$

Continued on next page



Table 8.17: ALUMODE 1101 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	0	1	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	0	$(P \wedge C \vee P \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	1	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	0	$(P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
0	1	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	0	$(P \wedge C \vee P \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg C \vee C \wedge \neg C)$
1	0	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	0	$(P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$

Continued on next page

Table 8.17: ALUMODE 1101 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	0	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
1	0	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	0	$(P \wedge C \vee P \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	1	0	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	0	$(P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	0	0	0	0	$(0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	0	1	$(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	0	$(P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	1	$(A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	1	0	0	$(0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	0	1	$(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	0	$(P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	1	$(A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	1	0	0	0	$(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	0	1	$(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	0	$(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	1	$(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	1	0	0	$(0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	0	1	$(PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	0	$(P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	1	$(A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

Table 8.18: ALUMODE 1110 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	0	1	$\neg(P \wedge 0 \vee P \wedge 0 \vee 0 \wedge 0)$

Continued on next page

Table 8.18: ALUMODE 1110 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y		X				
0	0	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge 0 \vee 0 \wedge 0)$
0	0	0	0	0	1	0	$\neg(0 \wedge PP2 \vee 0 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	0	$\neg(PP1 \wedge PP2 \vee PP1 \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$\neg(P \wedge PP2 \vee P \wedge 0 \vee PP2 \wedge 0)$
0	0	0	0	0	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge 0 \vee PP2 \wedge 0)$
0	0	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	0	1	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge 0 \vee 48'FFFFFFFFFFFFFF \wedge 0)$
0	0	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge 0 \vee C \wedge 0)$
0	0	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge 0 \vee C \wedge 0)$
0	0	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge PCIN \vee 0 \wedge PCIN)$
0	0	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge PCIN \vee PP2 \wedge PCIN)$
0	0	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge PCIN \vee 48'FFFFFFFFFFFFFF \wedge PCIN)$
0	0	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge PCIN \vee C \wedge PCIN)$
0	0	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge PCIN \vee C \wedge PCIN)$
0	1	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
0	1	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
0	1	0	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
0	1	0	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
0	1	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
0	1	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
0	1	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
0	1	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge P \vee C \wedge P)$
0	1	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
0	1	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge C \vee 0 \wedge C)$
0	1	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge C \vee 0 \wedge C)$
0	1	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge C \vee PP2 \wedge C)$
0	1	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge C \vee PP2 \wedge C)$
0	1	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge C \vee 48'FFFFFFFFFFFFFF \wedge C)$
0	1	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge C \vee C \wedge C)$
0	1	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge C \vee C \wedge C)$
0	1	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge C \vee C \wedge C)$
0	1	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge C \vee C \wedge C)$

Continued on next page

Table 8.18: ALUMODE 1110 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y		X				
1	0	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge P \vee 0 \wedge P)$
1	0	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge P \vee 0 \wedge P)$
1	0	0	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge P \vee PP2 \wedge P)$
1	0	0	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge P \vee PP2 \wedge P)$
1	0	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge P \vee 48'FFFFFFFFFFFFFF \wedge P)$
1	0	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge P \vee C \wedge P)$
1	0	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge P \vee C \wedge P)$
1	0	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge P \vee C \wedge P)$
1	0	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge P \vee C \wedge P)$
1	0	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge RS\_PCIN \vee 0 \wedge RS\_PCIN)$
1	0	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge RS\_PCIN \vee PP2 \wedge RS\_PCIN)$
1	0	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge RS\_PCIN)$
1	0	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	0	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge RS\_PCIN \vee C \wedge RS\_PCIN)$
1	1	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	0	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge RS\_P \vee 0 \wedge RS\_P)$
1	1	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge RS\_P \vee PP2 \wedge RS\_P)$
1	1	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge RS\_P \vee 48'FFFFFFFFFFFFFF \wedge RS\_P)$
1	1	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge RS\_P \vee C \wedge RS\_P)$

Continued on next page

Table 8.18: ALUMODE 1110 Observed Results (*cont.*)

OP Modes							Observed Outputs
Z		Y			X		
1	1	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge RS\_P \vee C \wedge RS\_P)$
1	1	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge RS\_P \vee C \wedge RS\_P)$

Table 8.19: ALUMODE 1111 Observed Results

OP Modes							Observed Outputs
Z	Y			X			
0	0	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge \neg 0 \vee 0 \wedge \neg 0)$
0	0	0	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge \neg 0 \vee PP2 \wedge \neg 0)$
0	0	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg 0 \vee 48'FFFFFFFFFFFFFF \wedge \neg 0)$
0	0	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge \neg 0 \vee C \wedge \neg 0)$
0	0	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge \neg PCIN \vee 0 \wedge \neg PCIN)$
0	0	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge \neg PCIN \vee PP2 \wedge \neg PCIN)$
0	0	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg PCIN)$
0	0	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	0	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge \neg PCIN \vee C \wedge \neg PCIN)$
0	1	0	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
0	1	0	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
0	1	0	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
0	1	0	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
0	1	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
0	1	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge \neg C \vee 0 \wedge \neg C)$

Continued on next page

Table 8.19: ALUMODE 1111 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
0	1	1	0	0	0	1	$\neg (PP1 \wedge 0 \vee PP1 \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	0	$\neg (P \wedge 0 \vee P \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	0	1	1	$\neg (A : B \wedge 0 \vee A : B \wedge \neg C \vee 0 \wedge \neg C)$
0	1	1	0	1	0	0	$\neg (0 \wedge PP2 \vee 0 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	0	1	$\neg (PP1 \wedge PP2 \vee PP1 \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	0	$\neg (P \wedge PP2 \vee P \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	0	1	1	1	$\neg (A : B \wedge PP2 \vee A : B \wedge \neg C \vee PP2 \wedge \neg C)$
0	1	1	1	0	0	0	$\neg (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	0	1	$\neg (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	0	$\neg (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	0	1	1	$\neg (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg C \vee 48'FFFFFFFFFFFFFF \wedge \neg C)$
0	1	1	1	1	0	0	$\neg (0 \wedge C \vee 0 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	0	1	$\neg (PP1 \wedge C \vee PP1 \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	0	$\neg (P \wedge C \vee P \wedge \neg C \vee C \wedge \neg C)$
0	1	1	1	1	1	1	$\neg (A : B \wedge C \vee A : B \wedge \neg C \vee C \wedge \neg C)$
1	0	0	0	0	0	0	$\neg (0 \wedge 0 \vee 0 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	0	1	$\neg (PP1 \wedge 0 \vee PP1 \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	0	$\neg (P \wedge 0 \vee P \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	0	1	1	$\neg (A : B \wedge 0 \vee A : B \wedge \neg P \vee 0 \wedge \neg P)$
1	0	0	0	1	0	0	$\neg (0 \wedge PP2 \vee 0 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	0	1	$\neg (PP1 \wedge PP2 \vee PP1 \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	0	$\neg (P \wedge PP2 \vee P \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	0	1	1	1	$\neg (A : B \wedge PP2 \vee A : B \wedge \neg P \vee PP2 \wedge \neg P)$
1	0	0	1	0	0	0	$\neg (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	0	1	$\neg (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	0	$\neg (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	0	1	1	$\neg (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg P \vee 48'FFFFFFFFFFFFFF \wedge \neg P)$
1	0	0	1	1	0	0	$\neg (0 \wedge C \vee 0 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	0	1	$\neg (PP1 \wedge C \vee PP1 \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	0	$\neg (P \wedge C \vee P \wedge \neg P \vee C \wedge \neg P)$
1	0	0	1	1	1	1	$\neg (A : B \wedge C \vee A : B \wedge \neg P \vee C \wedge \neg P)$
1	0	1	0	0	0	0	$\neg (0 \wedge 0 \vee 0 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	0	1	$\neg (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	0	$\neg (P \wedge 0 \vee P \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	0	1	1	$\neg (A : B \wedge 0 \vee A : B \wedge \neg RS\_PCIN \vee 0 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	0	$\neg (0 \wedge PP2 \vee 0 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	0	1	$\neg (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	0	$\neg (P \wedge PP2 \vee P \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	0	1	1	1	$\neg (A : B \wedge PP2 \vee A : B \wedge \neg RS\_PCIN \vee PP2 \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	0	$\neg (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	0	1	$\neg (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	0	$\neg (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	0	1	1	$\neg (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_PCIN \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	0	$\neg (0 \wedge C \vee 0 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	0	1	$\neg (PP1 \wedge C \vee PP1 \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	0	$\neg (P \wedge C \vee P \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	0	1	1	1	1	1	$\neg (A : B \wedge C \vee A : B \wedge \neg RS\_PCIN \vee C \wedge \neg RS\_PCIN)$
1	1	0	0	0	0	0	$\neg (0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	0	1	$\neg (PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	0	$\neg (P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	0	1	1	$\neg (A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	0	0	1	0	0	$\neg (0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	0	1	$\neg (PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	0	$\neg (P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	0	1	1	1	$\neg (A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	0	1	0	0	0	$\neg (0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	0	1	$\neg (PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	0	$\neg (P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	0	1	1	$\neg (A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	0	1	1	0	0	$\neg (0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	0	1	$\neg (PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

Continued on next page

Table 8.19: ALUMODE 1111 Observed Results (cont.)

OP Modes							Observed Outputs
Z	Y			X			
1	1	0	1	1	1	0	$\neg(P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	0	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	0	0	0	0	$\neg(0 \wedge 0 \vee 0 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	0	1	$\neg(PP1 \wedge 0 \vee PP1 \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	0	$\neg(P \wedge 0 \vee P \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	0	1	1	$\neg(A : B \wedge 0 \vee A : B \wedge \neg RS\_P \vee 0 \wedge \neg RS\_P)$
1	1	1	0	1	0	0	$\neg(0 \wedge PP2 \vee 0 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	0	1	$\neg(PP1 \wedge PP2 \vee PP1 \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	0	$\neg(P \wedge PP2 \vee P \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	0	1	1	1	$\neg(A : B \wedge PP2 \vee A : B \wedge \neg RS\_P \vee PP2 \wedge \neg RS\_P)$
1	1	1	1	0	0	0	$\neg(0 \wedge 48'FFFFFFFFFFFFFF \vee 0 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	0	1	$\neg(PP1 \wedge 48'FFFFFFFFFFFFFF \vee PP1 \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	0	$\neg(P \wedge 48'FFFFFFFFFFFFFF \vee P \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	0	1	1	$\neg(A : B \wedge 48'FFFFFFFFFFFFFF \vee A : B \wedge \neg RS\_P \vee 48'FFFFFFFFFFFFFF \wedge \neg RS\_P)$
1	1	1	1	1	0	0	$\neg(0 \wedge C \vee 0 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	0	1	$\neg(PP1 \wedge C \vee PP1 \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	0	$\neg(P \wedge C \vee P \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$
1	1	1	1	1	1	1	$\neg(A : B \wedge C \vee A : B \wedge \neg RS\_P \vee C \wedge \neg RS\_P)$

IRIS Test Article 4A Phase 1  
(TA4AP1)  
Datasheet  
Rev 2

August 13, 2012

University of Southern California  
Information Sciences Institute



## TABLE OF CONTENTS

---

1. Introduction .....	3
2. ITAGR1 Overview .....	4
2-1. Overall Test Article Architecture.....	4
3. Test Article Pinout.....	6
4. Electrical and Timing Information .....	7
5. Physical Chip Dimensions and Core Location.....	8

## 1. INTRODUCTION

---

This document represents the overall architecture of a test article designed at University of Southern California Information Sciences Institute for the DARPA IRIS program, Thrust Area 4a – Reliability in Digital ASICs. The test article contains a RISC processor connected through a point-to-point interconnect to an external memory interface. An overview and block diagram are presented for the test article, followed by references to other documents for further detail. A signal listing and physical die info are also provided.

## 2. ITAGR1 OVERVIEW

### 2-1. OVERALL TEST ARTICLE ARCHITECTURE

As noted above, this TA4AP1 test article (internal code name of itagr1) contains a RISC processor connected to an external memory interface through a point-to-point interconnect. The organization of the RISC processor with respect to the interconnect and the external memory interface is shown in Figure 1, while a depiction of the RISC processor is shown in Figure 2. The design of the RISC processor is similar to that of a design from the DARPA Trust in IC program that was called TA2 Software Article, with one notable exception. The memory interface of ITAGR1 has been redesigned to transform memory accesses into a burst of 32-bit transfers to reduce the pad/pin count of the resulting design. The point-to-point interconnect is implemented by the node bus interface (or memory interface) of each RISC processor. Besides serving as a controller for an external memory system, the external memory interface contains a node bus interface for interaction with the RISC processor. More detailed information about the subcomponents of ITAGR1 can be found in the accompanying documents *Test Article 2 Software Article RISC Processor Architecture Overview*, *Test Article 2 Software Article RISC Processor Instruction Set Manual*, and *Test Article 2 Software Article Memory Interface Description*.

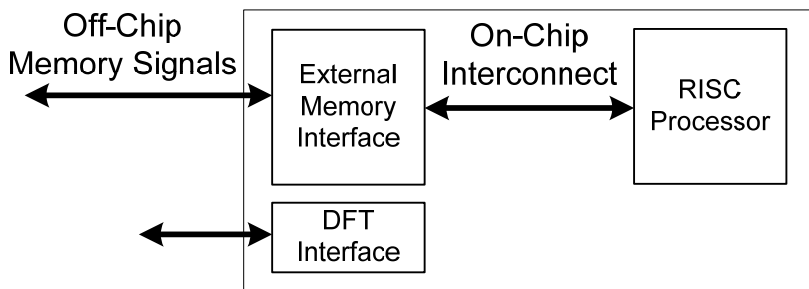


FIGURE 1 ITAGR1 ORGANIZATION

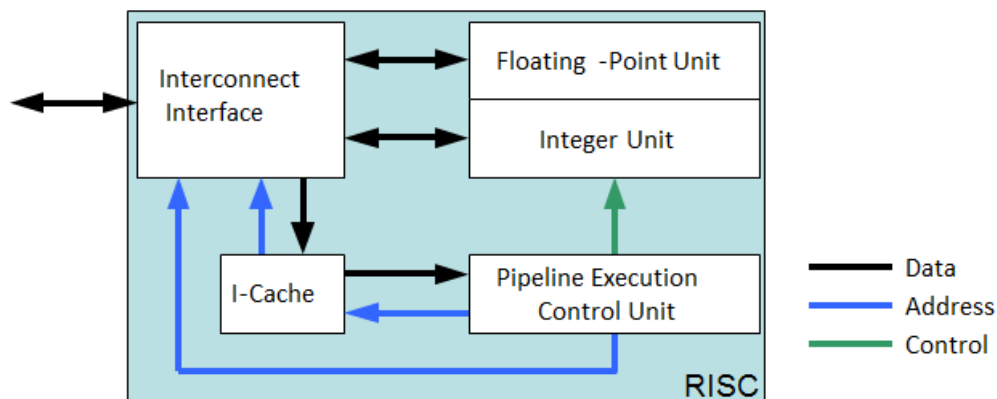


FIGURE 2 ITAGR1 RISC PROCESSOR ORGANIZATION

Since the only primary external interface concerns the external memory interface, almost all the significant signal I/O is associated with this interface. A listing of all signal I/O is as follows:

#### General I/O

input clk, reset, EMAA

#### External Memory Interface related I/O

input [31:0] edram\_do;  
output [31:0] edram\_di;  
output [7:0] edram\_bw;  
output [15:0] edram\_addr;  
output edram\_write\_enable\_n, edram\_read\_enable\_n;

#### Custom internal scan chain related I/O

input Scan\_I, Scan\_E  
output Scan\_O

#### JTAG boundary scan chain related I/O

input TCK, TRSTN, TDI, TMS  
output TDO

The EMAA input is a signal for fine-tune adjustment of the latency of the SRAM used for the instruction cache. The default value for this input is 0 (GND). For details, refer to the ARM memory compiler datasheets. It should also be noted that the edram\_bw signals are 32-bit word write enable signals for the memory interface. Every access through the memory interface is a 256-bit wide word that is serialized into a burst of eight 32-bit transfers.

### 3. TEST ARTICLE PINOUT

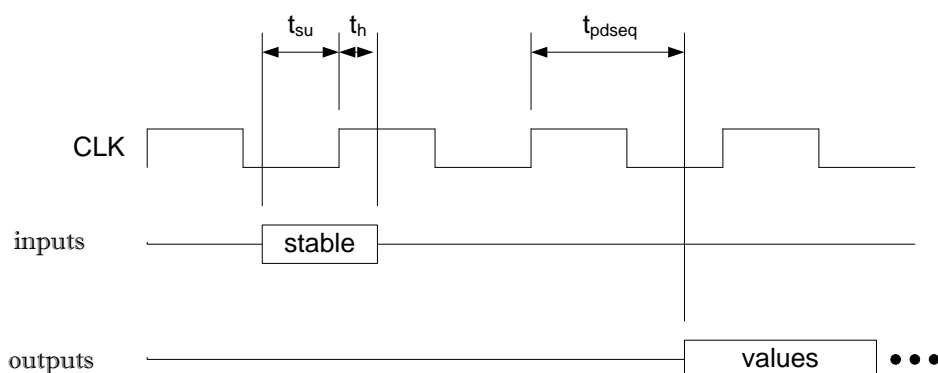
The table below lists the pad-to-signal assignment for the test article die as well as the pin-to-signal assignment for test articles that are bonded in PGA132M packages. Pad numbering is consistent with the MOSIS convention for this package; namely, pad 1 is the rightmost pad on the top edge of the chip, and numbering proceeds counter-clockwise. For more detail on this PGA132M package and bonding diagram numbering conventions, refer to documentation found at <http://www.mosis.com/Technical/Packaging/Ceramic/menu-pkg-ceramic.html>.

Pad Number / Bonding Finger	Pin	Signal Name	Signal Type	Pad Number / Bonding Finger	Pin	Signal Name	Signal Type	Pad Number / Bonding Finger	Pin	Signal Name	Signal Type
1	C3	padVDD	2.5V	45	N6	Scan_O	O	89	E14	edram_di_3	O
2	B1	padVSS	GND	46	P6	Scan_I	I	90	D14	edram_di_2	O
3	C2	edram_do_31	I	47	P7	Scan_E	I	91	E13	edram_di_1	O
4	D3	edram_do_30	I	48	N7	EMAA	I	92	C14	edram_di_0	O
5	C1	edram_do_29	I	49	M7	clk	I	93	D13	edram_bw_7	O
6	D2	edram_do_28	I	50	M8	reset	I	94	E12	padVDD	2.5V
7	D1	edram_do_27	I	51	N8	padVDD	2.5V	95	B14	padVSS	GND
8	E3	edram_do_26	I	52	P8	padVSS	GND	96	C13	edram_bw_6	O
9	E2	edram_do_25	I	53	P9	edram_di_31	O	97	D12	edram_bw_5	O
10	E1	edram_do_24	I	54	N9	edram_di_30	O	98	A14	edram_bw_4	O
11	F3	edram_do_23	I	55	M9	edram_di_29	O	99	B13	edram_bw_3	O
12	F2	edram_do_22	I	56	P10	edram_di_28	O	100	C12	edram_bw_2	O
13	F1	edram_do_21	I	57	P11	edram_di_27	O	101	A13	edram_bw_1	O
14	G1	edram_do_20	I	58	N10	edram_di_26	O	102	B12	edram_bw_0	O
15	G2	edram_do_19	I	59	P12	edram_di_25	O	103	C11	edram_addr_14	O
16	G3	coreVDD	1.0V	60	N11	edram_di_24	O	104	A12	padVDD	2.5V
17	H3	coreVSS	GND	61	M10	padVDD	2.5V	105	B11	padVSS	GND
18	H2	edram_do_18	I	62	P13	padVSS	GND	106	A11	edram_addr_15	O
19	H1	edram_do_17	I	63	N12	edram_di_23	O	107	C10	edram_addr_13	O
20	J1	edram_do_16	I	64	M11	edram_di_22	O	108	B10	edram_addr_12	O
21	J2	edram_do_15	I	65	P14	edram_di_21	O	109	A10	edram_addr_11	O
22	J3	edram_do_14	I	66	N13	edram_di_20	O	110	C9	edram_addr_10	O
23	K1	edram_do_13	I	67	M12	edram_di_19	O	111	B9	edram_addr_9	O
24	L1	edram_do_12	I	68	N14	edram_di_18	O	112	A9	edram_addr_8	O
25	K2	edram_do_11	I	69	M13	edram_di_17	O	113	A8	padVDD	2.5V
26	M1	edram_do_10	I	70	L12	edram_di_16	O	114	B8	padVSS	GND
27	L2	edram_do_9	I	71	M14	padVDD	2.5V	115	C8	edram_addr_7	O
28	K3	edram_do_8	I	72	L13	padVSS	GND	116	C7	edram_addr_6	O
29	N1	edram_do_7	I	73	L14	edram_di_15	O	117	B7	edram_addr_5	O
30	M2	edram_do_6	I	74	K12	edram_di_14	O	118	A7	edram_addr_4	O
31	L3	edram_do_5	I	75	K13	edram_di_13	O	119	A6	edram_addr_3	O
32	P1	padVDD	2.5V	76	K14	edram_di_12	O	120	B6	edram_addr_2	O
33	N2	padVSS	GND	77	J12	edram_di_11	O	121	C6	edram_addr_1	O
34	M3	edram_do_4	I	78	J13	edram_di_10	O	122	A5	edram_addr_0	O
35	P2	edram_do_3	I	79	J14	edram_di_9	O	123	A4	spare	NC
36	N3	edram_do_2	I	80	H14	edram_di_8	O	124	B5	TDO	O
37	M4	edram_do_1	I	81	H13	padVDD	2.5V	125	A3	coreVDD	1.0V
38	P3	padVDD	2.5V	82	H12	padVSS	GND	126	B4	coreVSS	GND
39	N4	padVSS	GND	83	G12	coreVDD	1.0V	127	C5	padVDD	2.5V
40	P4	coreVDD	1.0V	84	G13	coreVSS	GND	128	A2	padVSS	GND
41	M5	coreVSS	GND	85	G14	edram_di_7	O	129	B3	TDI	I
42	N5	edram_do_0	I	86	F14	edram_di_6	O	130	C4	TMS	I
43	P5	edram_write_enable_n	O	87	F13	edram_di_5	O	131	A1	TRSTN	I
44	M6	edram_read_enable_n	O	88	F12	edram_di_4	O	132	B2	TCK	I

## 4. ELECTRICAL AND TIMING INFORMATION

As shown in the previous section, TA4AP1 uses a core Vdd of 1.0V and a pad Vdd of 2.5V. Output pad drivers are rated at 9mA; thus, capacitive loading should be limited to around 10pF for reasonable slew rates on the output signals that will allow achieving the propagation delays shown below.

All timing information below is based on limited testing and simulation estimates. The worst-case conditions used during the testing were (T = room temperature, Vdd\_core = 0.97V, Vdd\_io = 2.25V). For these conditions, a clock period of 7.2ns was achieved in all cases. For simplicity, all inputs have been grouped together. While set-up times and hold times vary among inputs, the values listed below represent the worst-case values needed for correct operation.



Parameter	Value*
$t_{su}$	1ns*
$t_h$	5ns*
$t_{pdseq}$	7.2ns*

More detailed electrical and timing information will be added when available after reliability testing is conducted.

\* Chips have not been thoroughly tested for absolute input/output timing info. Depending on tester loads, input transition and output sampling times relative to the clock edge may require adjustment; however, a clock period of 7.2ns should be achievable in all cases.

## 5. PHYSICAL CHIP DIMENSIONS AND CORE LOCATION

The design submitted for fabrication was prepared with version V2.2.0.2IBM of the IBM 9SF PDK using the IBM 6\_02\_00\_00\_LB digital stack (for more info on this technology, refer to <http://www.mosis.com/ibm/9sf/>; note that the PDK DRC files may refer to this stack as 9SF\_6\_02\_00). The design as submitted measured 2.75mm x 2.75mm; however, with the inclusion of scribe lanes and other margins (refer to <http://www.mosis.com/products/assembly/#die-size> for examples), the fabricated die size may be somewhat larger. The fiducial provided by IBM contains marking identifiers in the lower left and upper right corners of the die and was included in the design file. Refer to the figure below, which shows relative locations of fiducial markings and the ITAGR1 chip core. The chip core, inside the pad ring, is roughly 1.57  $\mu\text{m}$  x 1.49  $\mu\text{m}$ . The table below provides x-y coordinate information for the points denoted in the figure. Note that each character in the fiducial lettering is comprised of multiple polygons.

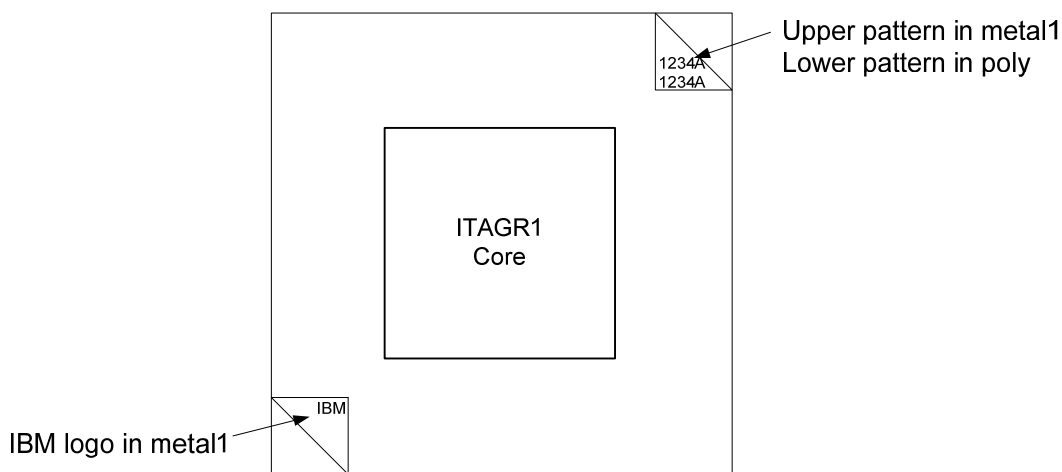


FIGURE 3 DEPICTION OF ITAGR1 DIE ORGANIZATION (NOT TO SCALE)

Point of Interest	Coordinates ( $\mu\text{m}$ )	
	x	y
Lower left corner of lower left polygon of the "1" in the metal1 "1234A" fiducial	2608.675	2637.675
Lower left corner of lower left polygon of the "1" in the polysilicon "1234A" fiducial	2608.675	2608.675
Upper right corner of upper right polygon of the "M" in the "IBM" fiducial	141.325	141.325
Lower left corner of ITAGR1 core	618.56	632.56

# IRIS Phase 2 Test Article (itagr1)

## Datasheet

March 3, 2014

University of Southern California  
Information Sciences Institute



## TABLE OF CONTENTS

---

1. Introduction .....	3
2. ITAGR1 Overview .....	4
2-1. Overall Test Article Architecture.....	4
3. Test Article Pinout.....	6
4. Electrical and Timing Information .....	7
5. Physical Chip Dimensions and Core Location.....	8

## 1. INTRODUCTION

---

This document represents the overall architecture of a digital test article designed at University of Southern California Information Sciences Institute for the DARPA IRIS program, Phase 2. The test article contains a RISC processor connected through a point-to-point interconnect to an external memory interface. An overview and block diagram are presented for the test article, followed by references to other documents for further detail. A signal listing and physical die info are also provided.

## 2. ITAGR1 OVERVIEW

### 2-1. OVERALL TEST ARTICLE ARCHITECTURE

As noted above, this IRIS Phase 2 digital test article (internal code name of itagr1) contains a RISC processor connected to an external memory interface through a point-to-point interconnect. The organization of the RISC processor with respect to the interconnect and the external memory interface is shown in Figure 1, while a depiction of the RISC processor is shown in Figure 2. The design of the RISC processor is similar to that of a design from the DARPA Trust in IC program that was called TA2 Software Article, with one notable exception. The memory interface of ITAGR1 has been redesigned to transform memory accesses into a burst of 32-bit transfers to reduce the pad/pin count of the resulting design. The point-to-point interconnect is implemented by the node bus interface (or interconnect interface), where one instance of the node bus interface resides in the RISC processor core and another in the external memory interface. More detailed information about the subcomponents of ITAGR1 can be found in the accompanying documents *Test Article 2 Software Article RISC Processor Architecture Overview*, *Test Article 2 Software Article RISC Processor Instruction Set Manual*, and *Test Article 2 Software Article Memory Interface Description*.

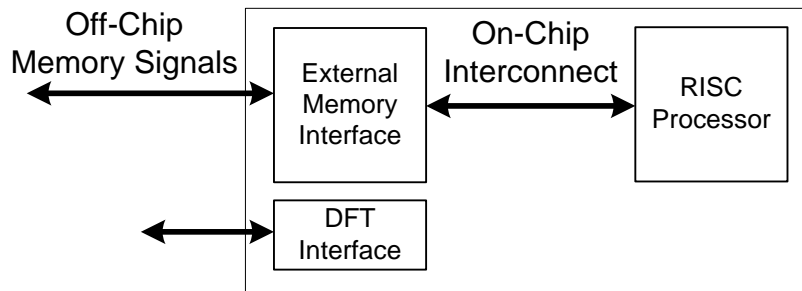


FIGURE 1 ITAGR1 ORGANIZATION

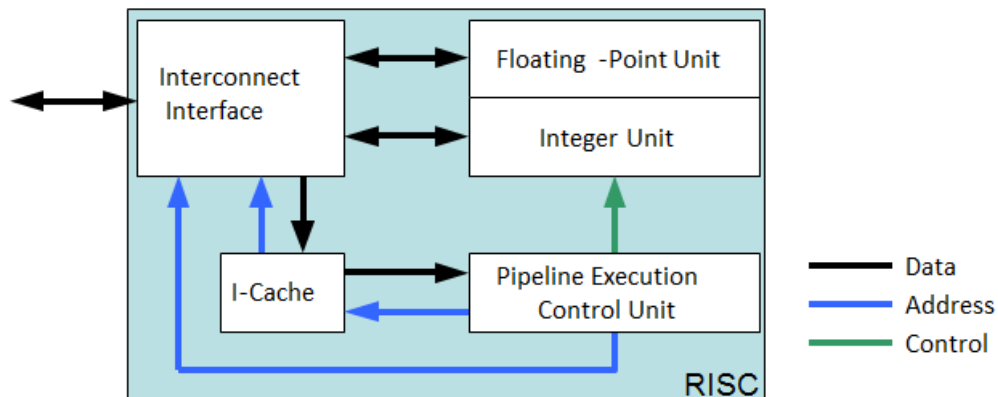


FIGURE 2 ITAGR1 RISC PROCESSOR ORGANIZATION

Since the only primary external interface concerns the external memory interface, almost all the significant signal I/O is associated with this interface. A listing of all signal I/O is as follows:

#### General I/O

```
input  clk, reset, EMAA
```

#### External Memory Interface related I/O

```
input [0:31] edram_do;  
output [0:31] edram_di;  
output [0:7] edram_bw;  
output [0:15] edram_addr;  
output edram_write_enable_n, edram_read_enable_n;
```

#### Custom internal scan chain related I/O

```
input Scan_I, Scan_E  
output Scan_O
```

#### JTAG boundary scan chain related I/O

```
input TCK, TRSTN, TDI, TMS  
output TDO
```

Note the big-endian labeling convention. The reset signal is an asserted-high synchronous reset. The EMAA input is a signal for fine-tune adjustment of the latency of the SRAM used for the instruction cache. The default value for this input is 0 (GND). For details, refer to the ARM memory compiler datasheets.

It should also be noted that every access through the memory interface is a 256-bit wide word that is serialized into a burst of eight 32-bit word transfers. The edram\_bw signals are word write enable signals for the memory interface, where each edram\_bw signal corresponds to a 32-bit word of the 256-bit wide word transfer. For detailed cycle-level timing information of the memory interface, refer to the companion representative test vector files.

### 3. TEST ARTICLE PINOUT

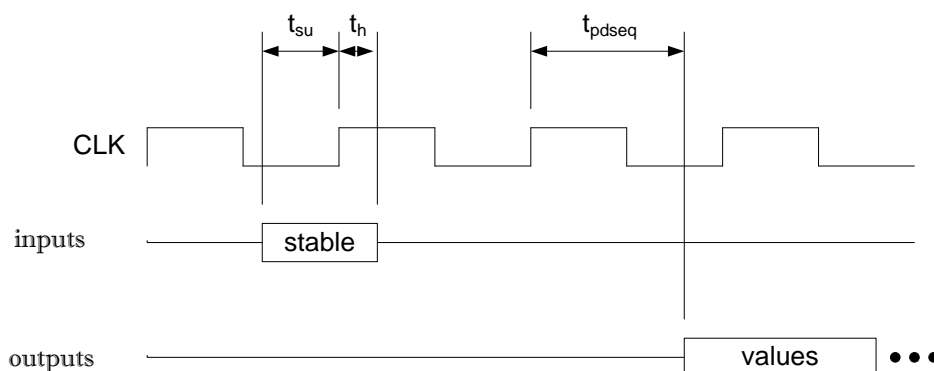
The table below lists the pad-to-signal assignment for the test article die as well as the pin-to-signal assignment for test articles that are bonded in PGA132M packages. Pad numbering is consistent with the MOSIS convention for this package; namely, pad 1 is the rightmost pad on the top edge of the chip, and numbering proceeds counter-clockwise. For more detail on this PGA132M package and bonding diagram numbering conventions, refer to documentation found at <http://www.mosis.com/Technical/Packaging/Ceramic/menu-pkg-ceramic.html>.

Pad Number / Bonding Finger	Pin	Signal Name	Signal Type	Pad Number / Bonding Finger	Pin	Signal Name	Signal Type	Pad Number / Bonding Finger	Pin	Signal Name	Signal Type
1	C3	padVDD	2.5V	45	N6	Scan_O	O	89	E14	edram_di_3	O
2	B1	padVSS	GND	46	P6	Scan_I	I	90	D14	edram_di_2	O
3	C2	edram_do_31	I	47	P7	Scan_E	I	91	E13	edram_di_1	O
4	D3	edram_do_30	I	48	N7	EMAA	I	92	C14	edram_di_0	O
5	C1	edram_do_29	I	49	M7	clk	I	93	D13	edram_bw_7	O
6	D2	edram_do_28	I	50	M8	reset	I	94	E12	padVDD	2.5V
7	D1	edram_do_27	I	51	N8	padVDD	2.5V	95	B14	padVSS	GND
8	E3	edram_do_26	I	52	P8	padVSS	GND	96	C13	edram_bw_6	O
9	E2	edram_do_25	I	53	P9	edram_di_31	O	97	D12	edram_bw_5	O
10	E1	edram_do_24	I	54	N9	edram_di_30	O	98	A14	edram_bw_4	O
11	F3	edram_do_23	I	55	M9	edram_di_29	O	99	B13	edram_bw_3	O
12	F2	edram_do_22	I	56	P10	edram_di_28	O	100	C12	edram_bw_2	O
13	F1	edram_do_21	I	57	P11	edram_di_27	O	101	A13	edram_bw_1	O
14	G1	edram_do_20	I	58	N10	edram_di_26	O	102	B12	edram_bw_0	O
15	G2	edram_do_19	I	59	P12	edram_di_25	O	103	C11	edram_addr_14	O
16	G3	coreVDD	1.0V	60	N11	edram_di_24	O	104	A12	padVDD	2.5V
17	H3	coreVSS	GND	61	M10	padVDD	2.5V	105	B11	padVSS	GND
18	H2	edram_do_18	I	62	P13	padVSS	GND	106	A11	edram_addr_15	O
19	H1	edram_do_17	I	63	N12	edram_di_23	O	107	C10	edram_addr_13	O
20	J1	edram_do_16	I	64	M11	edram_di_22	O	108	B10	edram_addr_12	O
21	J2	edram_do_15	I	65	P14	edram_di_21	O	109	A10	edram_addr_11	O
22	J3	edram_do_14	I	66	N13	edram_di_20	O	110	C9	edram_addr_10	O
23	K1	edram_do_13	I	67	M12	edram_di_19	O	111	B9	edram_addr_9	O
24	L1	edram_do_12	I	68	N14	edram_di_18	O	112	A9	edram_addr_8	O
25	K2	edram_do_11	I	69	M13	edram_di_17	O	113	A8	padVDD	2.5V
26	M1	edram_do_10	I	70	L12	edram_di_16	O	114	B8	padVSS	GND
27	L2	edram_do_9	I	71	M14	padVDD	2.5V	115	C8	edram_addr_7	O
28	K3	edram_do_8	I	72	L13	padVSS	GND	116	C7	edram_addr_6	O
29	N1	edram_do_7	I	73	L14	edram_di_15	O	117	B7	edram_addr_5	O
30	M2	edram_do_6	I	74	K12	edram_di_14	O	118	A7	edram_addr_4	O
31	L3	edram_do_5	I	75	K13	edram_di_13	O	119	A6	edram_addr_3	O
32	P1	padVDD	2.5V	76	K14	edram_di_12	O	120	B6	edram_addr_2	O
33	N2	padVSS	GND	77	J12	edram_di_11	O	121	C6	edram_addr_1	O
34	M3	edram_do_4	I	78	J13	edram_di_10	O	122	A5	edram_addr_0	O
35	P2	edram_do_3	I	79	J14	edram_di_9	O	123	A4	spare	NC
36	N3	edram_do_2	I	80	H14	edram_di_8	O	124	B5	TDO	O
37	M4	edram_do_1	I	81	H13	padVDD	2.5V	125	A3	coreVDD	1.0V
38	P3	padVDD	2.5V	82	H12	padVSS	GND	126	B4	coreVSS	GND
39	N4	padVSS	GND	83	G12	coreVDD	1.0V	127	C5	padVDD	2.5V
40	P4	coreVDD	1.0V	84	G13	coreVSS	GND	128	A2	padVSS	GND
41	M5	coreVSS	GND	85	G14	edram_di_7	O	129	B3	TDI	I
42	N5	edram_do_0	I	86	F14	edram_di_6	O	130	C4	TMS	I
43	P5	edram_write_enable_n	O	87	F13	edram_di_5	O	131	A1	TRSTN	I
44	M6	edram_read_enable_n	O	88	F12	edram_di_4	O	132	B2	TCK	I

## 4. ELECTRICAL AND TIMING INFORMATION

As shown in the pinout of the previous section, the IRIS Phase 2 digital test article nominally uses a core Vdd of 1.0V and a pad Vdd of 2.5V. Output pad drivers are rated at 9mA; thus, capacitive loading should be limited to around 10pF for reasonable slew rates on the output signals that will allow achieving the propagation delays shown below.

All timing information below is based on testing across a range of core voltages from 0.9V to 1.1V, I/O voltages from 2.25V to 2.75V, and temperatures from 0°C to 105°C. For these conditions, a clock period of 7.2ns was achieved in all cases. For simplicity, all inputs have been grouped together. While set-up times and hold times vary among inputs, the values listed below represent the worst-case values needed for correct operation.



Parameter	Value
$t_{su}$	2.5ns (min)
$t_h$	1ns (min)
$t_{pdseq}$	5ns (min_bc)* ; 9.6ns (min_wc)*

More detailed electrical and timing information may be added with program approval.

\* Depending on tester loads and testing conditions (bc=best case: Vcore=1.1V, Vpad=2.75V, temperature = 0°C; wc=worst case: Vcore=0.9V, Vpad=2.25V, temperature = 105°C), input transition and output sampling times relative to the clock edge may require adjustment; however, a clock period of 7.2ns should be achievable in all cases. For  $t_{pdseq}$  values exceeding a clock period, this indicates that the propagation delay for a specific output value corresponding to a particular clk cycle, as depicted in vcd test vector files, causes the output to not become valid until a following clk cycle; however, the chip will still run at the stated clock period but with outputs offset from their triggering clk edges by the stated  $t_{pdseq}$  values. Also, for output signal sampling at higher frequencies, some signal termination and VOH/VOL tuning may be necessary. For example, we found that terminating outputs through 50Ω into 0V and using VOH=VOL=0.6V provided best results on a Credence D10 tester.

## 5. PHYSICAL CHIP DIMENSIONS AND CORE LOCATION

The design submitted for fabrication was prepared with version V2.2.0.2IBM of the IBM 9SF PDK using the IBM 6\_02\_00\_00\_LB digital stack (for more info on this technology, refer to <http://www.mosis.com/ibm/9sf/>; note that the PDK DRC files may refer to this stack as 9SF\_6\_02\_00). The design as submitted measured 2.75mm x 2.75mm; however, with the inclusion of scribe lanes and other margins (for examples, refer to <http://www.mosis.com/pages/products/assembly/index#die-size>), the fabricated die size may be somewhat larger. The fiducial provided by IBM contains marking identifiers in the lower left and upper right corners of the die and was included in the design file. Refer to the figure below, which shows relative locations of fiducial markings and the ITAGR1 chip core. The chip core, inside the pad ring, is roughly  $1.57\text{ }\mu\text{m}$  x  $1.49\text{ }\mu\text{m}$ . The table below provides x-y coordinate information for the points denoted in the figure. Note that each character in the fiducial lettering is comprised of multiple polygons.

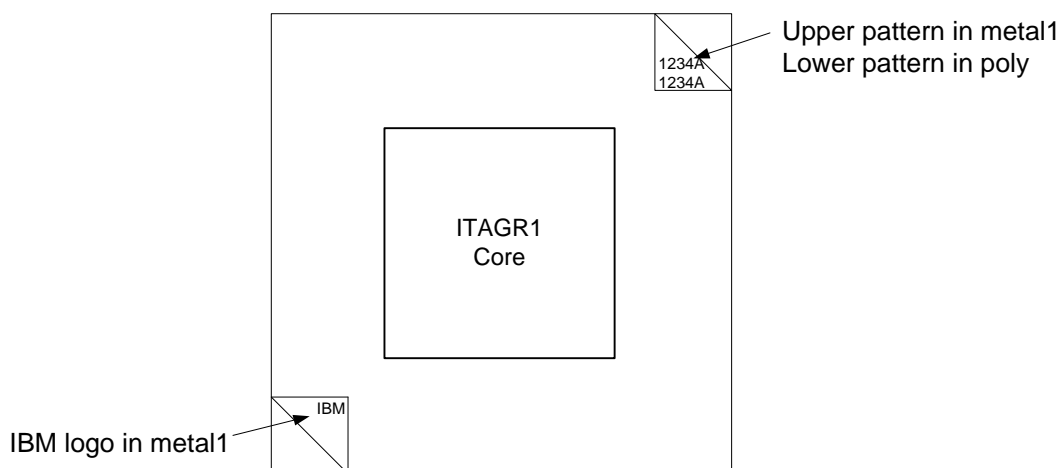


FIGURE 3 DEPICTION OF ITAGR1 DIE ORGANIZATION (NOT TO SCALE)

Point of Interest	Coordinates ( $\mu\text{m}$ )	
	x	y
Lower left corner of lower left polygon of the "1" in the metal1 "1234A" fiducial	2608.675	2637.675
Lower left corner of lower left polygon of the "1" in the polysilicon "1234A" fiducial	2608.675	2608.675
Upper right corner of upper right polygon of the "M" in the "IBM" fiducial	141.325	141.325
Lower left corner of ITAGR1 core	618.56	632.56

TEST ARTICLE 2 SOFTWARE  
(TA2\_SW)  
ARTICLE RISC Processor  
Architecture Overview

**University of Southern California  
Information Sciences Institute**

---



---

(This page intentionally left blank.)

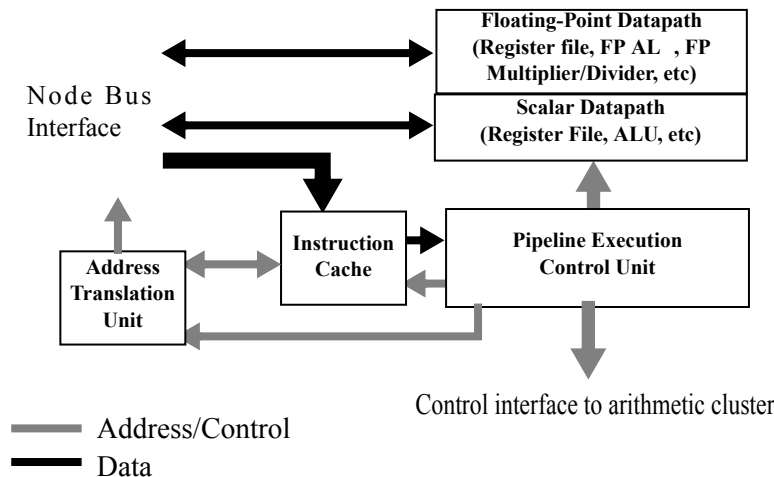
---

# Chapter 1 - Overview

---

## 1.0.1. Block Diagram

TheTA2\_SW node RISC processor executes threads (as opposed to streams) and supports single-issue, in-order execution, with 32-bit instructions and 32-bit addresses. A block diagram is shown in Figure 2. There are two internal datapaths: a *scalar datapath* that performs sequential operations on 32-bit integer operands and a *floating-point datapath* that performs operations on 32-bit single-precision floating-point operands. Additionally, the execution unit controls an external arithmetic cluster as a *wide datapath* that performs fine-grain parallel operations on 256-bit operands. This external wide datapath is a morphable unit that can operate independently as a streaming engine or under control of the RISC processor as a wide threaded processor. All datapaths execute from a single instruction stream under the control of a single 5-stage pipeline. The instruction set has been designed so that datapaths can, for the most part, use the same opcodes and condition codes, generating a large functional overlap. Each datapath has its own independent register file, but special instructions permit direct transfers between register files without going through memory.



**Figure 1: VC4aUY RISC Processor Architecture**

The combination of the execution control pipeline, scalar datapath, and floating-point datapath may be viewed as a conventional microprocessor and may be programmed as such. This capability is essential to an evolutionary software development approach. Users may, with very little effort, exploit coarse-grain parallelism by simply programming multiple nodes in a conventional sense. However, users may also exploit fine-grain parallelism by using the external arithmetic cluster as a wide datapath.

In addition to the execution unit and datapaths, eachTA2\_SW RISC processor includes other units of note. A small *instruction cache (IC)* is used to keep instruction accesses to the memory macro from interfering with data accesses as much as possible. A segment-based *address translation unit (ATU)* for converting virtual to physical addresses is also incorporated into the RISC processor.

---

### 1.0.2. Scalar Datapath and Execution Pipeline

The execution pipeline is shared between the scalar, floating-point and wide datapaths. It is a standard 5-stage pipeline, with the following stages: (1) instruction fetch; (2) instruction decode and register read; (3) execute; (4) memory; and, (5) register writeback. There are a number of events which cause pipeline hazards, for example: (1) long instruction sequences, such as for multiplies and divides; (2) register operations, involving data dependences between nearby instructions; and, (3) memory operations, which stall the pipeline due to multiple cycles latency to memory. The second class of hazards usually incur no extra latency penalty due to the incorporation of register forwarding. Other hazards can only be resolved through pipeline stalls or avoided through careful ordering of instructions by the compiler.

The scalar datapath is for the most part a standard RISC architecture that supports both supervisor and user level processing, augmented with a few TA2\_SW-specific functions for coordinating with the wide datapath. Scalar register values are used for addressing operations, as well as for controlling subfield operations

### 1.0.3. Floating-Point Datapath

The floating-point (FP) datapath implements a subset of the IEEE-754 floating-point standard. Since target applications are mostly from the embedded signal processing realm, only single-precision numbers are supported. To achieve a better area-performance solution, operations on denormalized numbers are not supported and cause exceptions. In addition, whenever a result is a denormalized number, an underflow exception is raised and the minimum normalized number is produced for output. The inexact exception flag on division operations is not IEEE-754 compliant, which is common for multiplicative division algorithms. Additional operations are necessary to correct this. Other exception flags – Invalid, Divide by Zero, Overflow, Underflow and Inexact (except divide) – are accurately generated as specified by the IEEE-754 standard. All four rounding modes are implemented.

The floating-point datapath is under control of the same execution pipeline that controls the integer scalar datapath. Since floating-point operations require multiple execution cycles in the execute stage, FP instruction completion latency is larger than that for integer instructions. However, since the FP datapath is pipelined, a throughput of one instruction per clock cycle can be achieved in most cases as long as there are no data dependences between co-existing instructions in the FP pipeline. The only exception is for the divide instruction which reuses FP pipeline stages during its execution.

### 1.0.4. Wide Datapath

When controlled by the RISC processor as a wide datapath, the arithmetic cluster processes objects aggregated within a row of the local memory array by operating on 256 bits in a single processor cycle. This fine-grain parallelism offers additional opportunity for exploiting increased processor-memory bandwidth available in an embedded DRAM design. The WideWord unit can perform bit-level operations, such as simple pattern matching, or higher-order computations such as searches and reduction operations.

The WideWord datapath has several features to distinguish it from a conventional SIMD architecture. First is the ability to **change ALU operand width** on a per-instruction basis, enabling it to treat a 256-bit value as a packed array of objects of eight, sixteen, or thirty-two bits in size. This characteristic means the WideWord ALU is more accurately represented as parallel ALUs, where the number of ALUs depends on the operand size. Second, a **permutation network** enables applications to rapidly align and reorganize wide register operands. Third, it supports **selective execution** of instructions on sub-fields within a WideWord, depending on the state of local and neighboring condition codes. Fourth, even for applications where the WideWord ALU operations are not applicable, the wide datapath can be used to **accelerate memory access time and communication**.

### 1.0.5. Instruction Cache

A small instruction cache is included to avoid instruction accesses interfering with data requests, both to reduce the frequency of requests to memory and to maximize the opportunity for faster page mode accesses for the data requests. The instruction cache is direct mapped, and the size for the initial implementation is 4Kbytes with 32byte cache lines. Because it caches just instructions, which are not expected to be modified during program execution, there is no write back facility or other mechanisms for keeping cache lines coherent with memory. To support context switching, an invalidate instruction permits invalidation of individual cache lines.

---

### 1.0.6. Address Translation

The address translation scheme employed by TA2\_SW uses *segments*, each of which is defined by segment registers containing a physical base address and limit. The local memory region is partitioned into eight segments at fixed virtual bases, for kernel code, stack and data, user code and data/stack, and for kernel and user communication buffers. A small number of global segment registers are also used; since global segments must be able to map portions of a shared virtual address space much larger than the physical memory of an individual node, global segments must be represented by both a virtual and physical base address register. Device IDs are included in the translation registers to support the TA2\_SW node interconnect specification.

Remote addresses are translated via the concept of a home node, which is guaranteed to have the translation. Therefore, a node must maintain translation information for only eight local segments plus a small number of segments for its portion of the global memory, as well as for any remote data for which it is the home node. The major advantages of this approach are that translation may be accomplished rapidly, and translation information on each node scales well.

## 1.1. Other Features

### 1.1.1. Exceptions

Exceptions, arising from execution of node instructions, and interrupts, from other sources such as an internal timer or external interrupt signal, are handled by a common mechanism. The exception handling scheme for TA2\_SW has a modest hardware requirement, exporting much of the complexity to software, to maintain a flexible implementation platform. It provides an integrated mechanism for handling hardware and software exception sources. Additionally, it provides a flexible priority assignment scheme which minimizes the amount of time that exception recognition is disabled. While the hardware design supports traditional stack-based exception handlers, we also outline a non-recursive dispatching scheme which uses hardware features to allow preemption of lower-priority exception handlers using a mechanism which should be easier to debug.

The remainder of this document is organized as follows. Chapter 2 describes the registers and data types used in the TA2\_SW RISC processor. Chapter 3 gives an overview of the instruction set architecture (ISA) followed by a description of the execution pipeline and scalar datapath in Chapter 4. X presents an overview of the floating-point datapath, while Chapter 6 describes some of the more interesting features of the arithmetic cluster when controlled by the RISC processor as a wide datapath. Finally, Chapter 7, Chapter 8, and Chapter 9 present brief descriptions of the instruction cache, address translation, and exceptions, respectively.

---

## Chapter 2 - Registers and Data Types

---

### 2.1. Introduction

This chapter describes TA2\_SW's different registers and their usages, and how data is represented in these registers. The scalar, floating point, and wide datapaths each have their own register file. Whether an instruction uses the scalar, floating-point or wide datapath, arithmetic operations generally follow a 3-register format, with two sources and one destination. Transfers between register file is accomplished with explicit move instructions. Data is transferred between memory and registers with explicit load and store instructions only. Memory operations involving scalar, floating-point, and wide registers refer to memory locations aligned at 32-bit, 32-bit, and 256-bit boundaries, respectively.

The general-purpose registers, including scalar integer, floating-point, and wide, can be accessed in either user mode or supervisor mode. Some special-purpose registers can be accessed in user mode, but all remaining special-purpose registers may be accessed only in supervisor mode. For the most part, the registers in the scalar integer and floating-point datapaths follow standard RISC systems. The wide datapath, in contrast, has several novel types of registers to facilitate selective execution on specific subfields of the register. The condition codes have been extended on the wide datapath to maintain a result for each separate data field and branch instructions have been added to the ISA to simultaneously check the conditions on all data fields. Another novel feature of the wide datapath is the ability to select an individual subfield of the wide register, using either an immediate or a scalar general-purpose register, and move the selected field in an explicit move instruction.

Beyond the standard supervisor-level registers required for interrupts, exceptions and protection, a few special-purpose registers in the system support TA2\_SW-specific activities. Segment registers are used to support address translation. Also, an environment identifier (EID) identifies the currently active user program, for protection purposes.

### 2.2. Description of Node Registers

The registers for a TA2\_SW node are summarized in Table 1 and graphically displayed in Figure 4. This section describes each type of register in detail. In the classification below, we first describe the general-purpose registers, then the special-purpose registers, distinguishing between supervisor-level registers and user-level registers. Access privileges are described by the mode field of the program status word (PSW) register. This organization is also reflected in Table 1 and Figure 4. In Table 1, the "type" field describes the classification of each register. Type *scalar*, *floating-point*, and *WideWord* refer to the general-purpose registers, *SP* indicates the user-level special-purpose registers, *AT* refers to the address translation registers, and *P* refers to all other privileged registers.

This section describes the general-purpose scalar and wide registers that are accessible to user code.

#### 2.2.0.1. General-Purpose Scalar Registers

There are 32 general-purpose scalar registers, each 32-bits wide, which we designate as R0-R31 in Figure 4. This register file is used as the source or destination for all integer scalar instructions. In addition, scalar registers are used to provide addresses for memory accesses to scalar and wide load/store instructions. Further, scalar general-purpose registers can be used to index subfield in a wide register during transfers between register file using the MVSUI and MVWSI instructions (see below). Memory operations to load and store objects to/from a general-purpose scalar register are aligned at 32-bit boundaries. For convenience in performing arithmetic operations where the immediate 0 is one of the operands, R0 is hardwired to hold the value 0.

---

### 2.2.0.2. Floating-Point Registers

There are 32 general-purpose single-precision floating-point registers, each 32-bits wide, which we designate as FR0-FR31 in Figure 4. This register file is used as the source or destination for all scalar floating-point instructions. Floating-point register values can be transferred to/from scalar or wide datapaths via special transfer instructions (see ISA manual for details). Memory operations to load and store objects to/from a general-purpose floating-point register are aligned at 32-bit boundaries.

### 2.2.0.3. General-Purpose Wide Registers

There are 32 general-purpose wide registers, each 272-bits wide representing 256 bits of data and 16 bits of token, which we designate as WR0-WR31 in Figure 4. This register file is used as the source or destination of all wide instructions. Wide instructions perform the same operation on 8-, 16-, or 32-bit subfields of the wide register, as designated by the width (WW) field of the instruction (Future implementations may also support 64-bit subfields for wide double-precision floating point capability.) The mask register and participation mode register (described below) can optionally be used to designate which subfield will participate in an instruction, if the participation (PP) field of the instruction is set.

Each entry of the WideWord register file contains not only 256 bits of data, but also 16 bits of token information, 2 bits for each 32 bits of data, as is consistent with the association of tokens and data in the TA2\_SW streaming operations executed in the arithmetic clusters when configured for streaming mode (refer to the specification for the arithmetic cluster). The rationale for including tokens in the WideWord datapath is that the WideWord unit may be involved in processing streams stored in memory, and it is desirable for the tokens of the stream to be preserved for future streaming operations. To support this capability, nominally the tokens associated with the operand specified by wrA are written to the token field of the operand specified by wrD in any WideWord instruction. However, some TA2\_SW implementations may ensure token compliance for only WLD and WST instructions. For designs that implement the full token capability, tokens are not subject to selective execution. That is, the tokens of wrA will be written to wrD even if the participation effect masks off all data field of wrD.

Wide registers are loaded from/stored to memory using addresses from the general-purpose scalar registers. Memory operations to load/store objects to/from a general-purpose wide register are aligned at 256-bit boundaries. Individual field of wide word registers can also be set or read using MVSW, MVWS, MVSWI and MVWSI instructions that use a register or immediate index to specify the data field to be accessed. In addition to arithmetic and transfer operations, wide registers can be updated through the permutation instructions WPRM and WPRMI, which reorganize the data field of the source register into a destination register. The former instruction uses a third wide register to specify how the data fields will be rearranged, and the latter performs a lookup into a table of hardcoded permutation patterns.

### 2.2.1. User-Level Special-Purpose Registers

A large number of special-purpose registers are directly or indirectly accessible to the user program, each described in this section.

- A single condition register for scalar condition codes, and a set of five condition registers for wide condition codes
- Scratch registers for scalar integer multiply and divide
- A participation mode register and mask register to support selective execution on the wide ALU

In addition to being read/written indirectly by other ALU operations, the architecture permits user-level access to any special-purpose register through explicit moves to standard registers, using the MTSPR and MFSPR instructions.

---

### 2.2.1.1. Scalar Condition Register

The scalar condition code register, CC in Figure 4, consists of 5 bits. The first three bits of CC are set by an algebraic comparison of the result to zero; the other two bits have slightly more peculiar semantics. The condition codes have the CC bit-labels and semantics as indicated in Figure 2. Note that LT, GT, EQ, and CA condition codes are updated only if the current instruction has its condition code enable bit set. The OV condition code is updated for any scalar add or subtract operation, regardless of the condition code enable bit setting, and is sticky; that is, it is only cleared when the condition code register is read. They are accessed in conditional branch and call statements. Further, like any user-level special-purpose registers, they can be explicitly read and written with the MFSPR and MTSPR instructions, respectively. When accessed with these instructions, the 5-bit CC value is right-justified to the least significant bits of the 32-bit integer datapath.

Condition Code	CC bit	Description
LT	0	This bit is set when the result represents a number strictly less than zero.
GT	1	This bit is set when the result represents a number strictly greater than zero.
EQ	2	This bit is set when the result represents a number equal to zero.
OV	3	This bit is set to indicate overflow has occurred during execution of an add or subtract instruction. This bit is not altered by any other instructions. In practice, the OV bit is set if the carry out of bit 0 is not equal to the carry out of bit 1 (assuming big Endian bit labeling).
CA	4	In general, the carry bit (CA) is set to indicate that a carry out of bit 0 occurred during execution of an add or subtract instruction. This bit is not altered by any other instructions.

**Figure 2: Scalar Condition Code Register**

### 2.2.1.2. Wide Condition Registers

While the scalar codes are consolidated into a single condition register, the CC described above, each type of WideWord condition code is allocated an entire register so the results of parallel operations on objects as small as bytes may be recorded. Each one of these condition registers is 32 bits wide. Thus, wide condition registers are designated as LT, GT, EQ, OV, and CA. For an example of how the wide condition registers are used, a bit of the WideWord LT register is set if the result of its corresponding 8-bit datapath is negative. However, there are subtleties due to the configurability of the operand sizes. For example, if a WideWord instruction specifies that operands are to be treated as 32-bit values, the condition codes are grouped into eight groups of 4, where each bit of a group is updated with the same value to reflect a condition for the group's corresponding 32-bit result. Like the scalar CC register, the LT, GT, EQ, and CA wide condition registers are only set by instructions that have their C field enabled. The OV register is a sticky register that is updated on all WideWord add and subtract operations; bits of this register are cleared only when the register is read using an MFSPR instruction.

The wide condition codes are accessed by the branch instructions BAX and BNx, which represent Branch-On-All and Branch-On-None conditions for the appropriate wide condition register represented by x.

Similar to condition codes, the WideWord floating-point status register (FPSR - special-purpose register 15) may be updated to reflect exception conditions for WideWord floating-point operations. This register is a 32-bit register arranged in groups of 4 status conditions for each of the eight 32-bit floating-point units in the WideWord datapath. The 4 status conditions are: invalid (IV), inexact (IX), overflow (OV), and underflow (UD). Refer to the IEEE-754 standard for details. All bits of FPSR are sticky; once set, they remain set until FPSR is read via an MFSPR instruction. The bit arrangement for FPSR is shown below.



Two registers, designated HI and LO in Figure 4, are automatically set as the result of a scalar integer multiply or divide. HI holds the most significant 32 bits of a multiplication result or the quotient of a division. LO has the least significant 32 bits of a multiplication result or the remainder of a division.

The Participation Mode (PM) register is a 5-bit register that describes the conditions for selective execution of a wide instruction that has its PP field set. The conditions correspond to the four condition codes or the mask register M (as will be discussed in Chapter 6). The PM register is read/written using the MFSPR and MTSPR instructions. When accessed with these instructions, the 5-bit PM value is right-justified to the least significant bits of the 32-bit integer datapath. It is also updated automatically to select the Mask Register (M) for participation when M is updated.

The mask register is a 32-bit register used in participation, which we refer to as M in Figure 4. If the PP field of a wide instruction is set, and the M bit of the PM register is set, then the instruction is conditionally executed on each data field that has its corresponding bit in the M register set. Like the WideWord condition codes, if the width of each field is larger than 8 bits, multiple bits in the M register will be set corresponding to a single data field (2 for 16-bit widths, 4 for 32-bit widths). Update of the M register automatically causes the M bit of the PM register to be set.

A total of 28 32-bit registers related to local and global segments are used to perform translation of virtual addresses to physical addresses by the node processor. A detailed description of how these registers are used in the address translation process can be found in Chapter 8. The registers are set by supervisor-level software using MTPR instructions, usually as a result of a context switch or a change in the size or location of current global segments. They are read either by MFPR instructions, or more commonly, directly by address translation hardware.



---

### 2.2.3. Other Supervisor-Level Registers

A set of 16 registers support local segments, referring to addresses local to the node that are inaccessible to other nodes. There are eight local segments, with two registers representing each segment. The Local Segment Base registers (SB0-SB7) hold the physical base address of each local segment. The Local Segment Limit registers (SL0-SL7) hold the maximum offset from the base, for address bounds checking, as well as some additional bits to support access protection.

A set of 12 registers support global segments, referring to addresses that may be shared across nodes. There are four global segments, and each is supported by three separate registers. Global segments must be able to map portions of a shared virtual address space much larger than the physical memory of an individual node. For this reason, global segments have both Global Segment Physical Base registers (GPB0-GPB3), similar to local segments, as well as Global Segment Virtual Base Registers (GVB0-GVB3). Usages of the Global Segment Limit registers (GL0-GL3) are analogous to the SL0-SL7 registers for local segments.

A number of other supervisor-level registers are included to support the run-time kernel activities. These can be classified into the following categories:

- Scratch registers
- The program counter
- The processor status word
- The environment identifier
- Timer registers, including two to hold current system clock and one used as a countdown timer
- Registers to support interrupts and exceptions, a total of seven

While in some cases these registers are updated as a result of a hardware event or upon execution of some other instruction, all of the registers can be read from/written to general-purpose registers by the supervisor-level instructions MFPR and MTPR. There are two exceptions to this. The Program Counter is set only by hardware, and cannot be accessed directly, even by supervisor-level code; for this reason, it is not given a register class in Table 1. Also, bits of the Exception Source Word (ESW) are set or cleared in software only indirectly through the Exception Set Register and the Exception Reset Register, respectively, although it can be read by MFPR; MTPR to the ESW is undefined and is treated as a no-op by the hardware.

#### 2.2.3.1. Scratch registers

Four 32-bit scratch registers, designated SCR0-SCR3 in Figure 4, are used by the kernel for its various activities. The goal of having these additional registers is to avoid the need to save and restore context of general-purpose registers when switching between the kernel and user-level code. The kernel can instead copy the contents of up to four of the general-purpose registers into SR0-SR3, then use the general-purpose registers, and subsequently restore the contents of the general-purpose registers, thus avoiding more costly memory accesses.

#### 2.2.3.2. Program counter

The program counter (PC) maintains the address to the current instruction to be executed. Although user code causes the PC register to be updated, it is updated indirectly through the execution of instructions that change the flow of control in the program (*i.e.*, branches, procedure calls and interrupts and exceptions).

Upon execution of a branch instruction, the PC is updated by hardware to the target of the branch. For a CALL instruction, the current PC is copied into SR31, and then the PC is updated to the starting point of the called function. A subsequent RET instruction will cause R31 to be

---

copied back to PC. On an interrupt or exception, the current PC is automatically copied into the FADR register (see description below), and is restored from FADR upon execution of a RFE instruction.

#### **2.2.3.3. Processor status word**

The processor status word is shown as PSW in Table 8. A detailed description of the PSW and its operation is given in Chapter 9.

#### **2.2.3.4. Environment identifier**

A 16-bit EID register records the currently active user context, and it is used to support communication between nodes. The EID register is set by the kernel upon context switch. When accessed with MTPR or MFPR instructions, the 16-bit EID value is right-justified to the least significant bits of the 32-bit integer datapath.

#### **2.2.3.5. Timer registers**

Two 32-bit registers, RCL and RCH, hold the low-order and high-order bits, respectively, of the real-time clock. The real-time clock provides a high-resolution measure of real time for indicating the time of day and date. The combination of RCL and RCH may be viewed as a loadable 64-bit counter. At reset, the value of RCH and RCL are all 0s and begin incrementing when reset is released. The real-time clock is clocked by the CPU clock. Considering a probable CPU frequency range of 200MHz to 1GHz for implementations over the life of this architecture, the real-time clock will provide ranges of approximately 117 to 585 years at a 1ns to 5ns resolution, respectively. RCH and RCL values may be initialized to desired values through the use of the MTPR instruction and are read using the MFPR instruction.

The TIMER register is a 32-bit decrementing counter that provides a mechanism for causing an interrupt after a programmable delay. The frequency of the TIMER decrement is the same as the CPU clock frequency. The TIMER causes an exception (subject to masking) when it reaches 0 and begins immediately to count down the next interval without processor intervention. The interval is set by loading the TIMER register with the interval value by initially using an MTPR instruction. Subsequently, the TIMER returns to the interval value the next cycle after counting down to a 0 value.

#### **2.2.3.6. Registers to support interrupts and exceptions**

There are eight 32-bit registers, shown in Figure 4, that are used to support interrupts and exceptions. A detailed description of their usage can be found in Chapter 9.

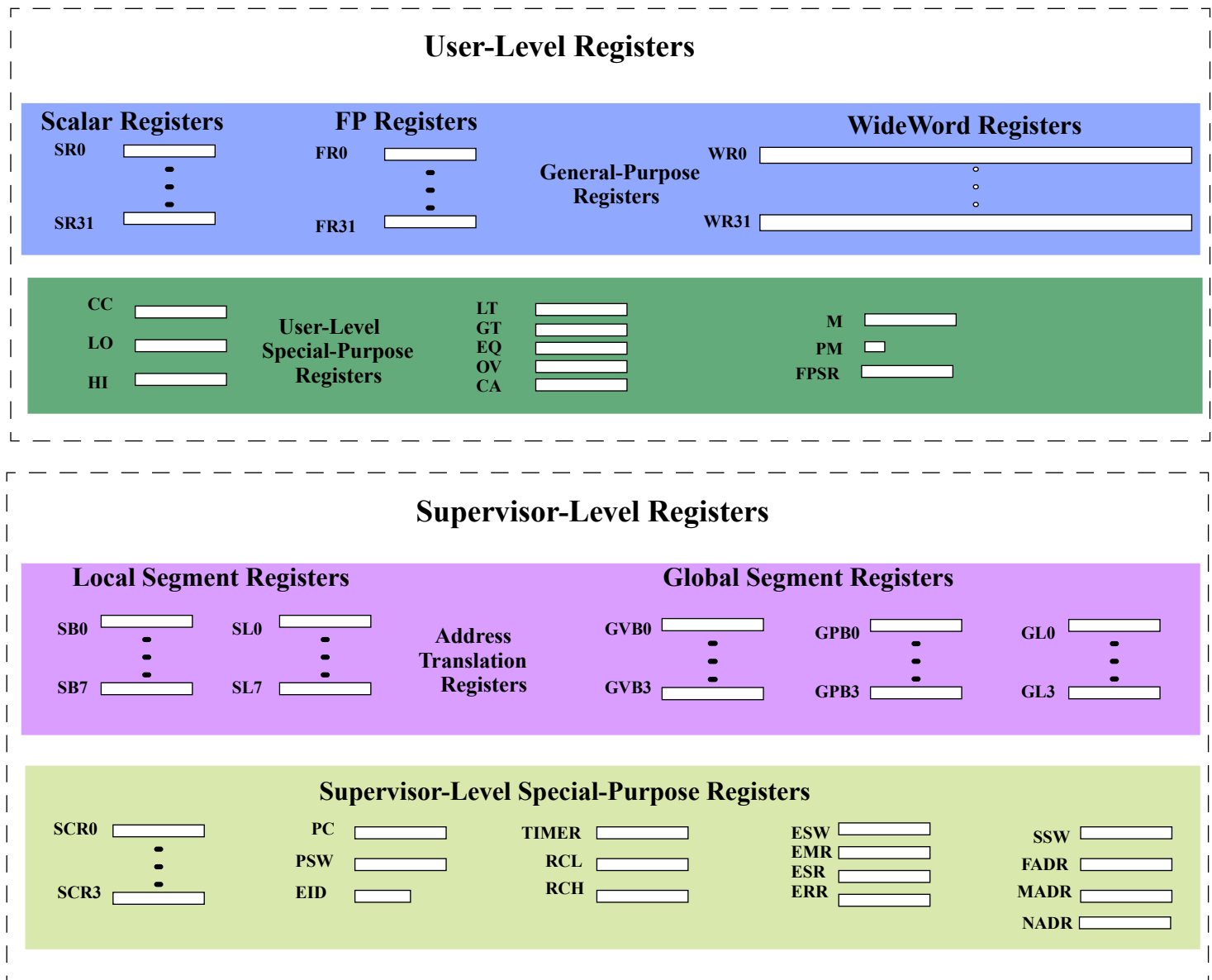
The Stored PSW register (SSW) holds the value of the PSW immediately prior to the interrupt or exception. The MADR and FADR registers hold the address of the faulting memory address and/or faulting instruction, in the event of an exception. If the cause of the exception was just a normal timer-initiated interrupt, the FADR register will hold the next instruction to be executed. The NADR holds the address of the instruction that was issued after that pointed to by the FADR. This value is useful when recovering from exceptions that occur while branches are in the pipeline. All of these registers are set either by hardware in the event of a hardware exception, or by MTPR instructions at the beginning of a software exception. The PC and PSW registers are restored with the values of FADR and SSW, respectively, on execution of a RFE instruction.

The four additional registers to support exceptions are the Exception Enable Mask register (EMR), the Exception Source Word (ESW), the Exception Set register (ESR) and the Exception Reset register (ERR). The EMR register indicates which exceptions are currently enabled, and is set by the supervisor. Fields of the ESW are set to 1 either directly by hardware in the event of a hardware exception, or by software

setting corresponding bits in the ESR register for software exceptions. Bits of the ESW are cleared to 0 by software setting corresponding bits in the ERR register. A description of the bit fields and their meaning can be found in Chapter 9

NAME	Type	Number	Width	DESCRIPTION
<b>R0-R31</b>	scalar	0 - 31	32	General-purpose scalar integer registers
<b>FR0-FR31</b>	floating-poin	0 - 31	32	General-purpose scalar floating-point registers
<b>WR0-WR31</b>	WideWord	0 - 31	272	General-purpose WideWord registers (256-bit data plus 16-bit token)
<b>CC</b>	SP	0	5	LT, GT, EQ, OV, and CA bits of scalar processor
<b>HI</b>	SP	1	32	most significant 32 bits of multiplication result, quotient of division
<b>LO</b>	SP	2	32	least significant 32 bits of multiplication result, remainder of division
<b>LT</b>	SP	8	32	Less Than condition code register of WideWord Unit
<b>GT</b>	SP	9	32	Greater Than condition code register of WideWord Unit
<b>EQ</b>	SP	10	32	Equal condition code register of WideWord Unit
<b>CA</b>	SP	11	32	Carry condition code register of WideWord Unit
<b>OV</b>	SP	12	32	Overflow condition code register of WideWord Unit
<b>M</b>	SP	13	32	WideWord Mask register used in selective execution
<b>PM</b>	SP	14	5	WideWord Participation Mode register used in selective execution
<b>FPSR</b>	SP	15	32	WideWord floating-point status register
<b>SB0-SB7</b>	AT	0 - 7	32	Base registers for local segments, used for address translation
<b>SL0-SL7</b>	AT	8 - 15	32	Limit registers for local segments, used for address translation
<b>GVB0-GVB3</b>	AT	16 - 19	32	Virtual base registers for global segments, used for address translation
<b>GL0-GL3</b>	AT	20 - 23	32	Limit registers for global segments, used for address translation
<b>GPB0-GPB3</b>	AT	24 - 27	32	Physical base registers for global segments, used for address translation
<b>PSW</b>	P	0	32	Processor status word
<b>SSW</b>	P	1	32	Stored value of PSW, used in exception handling
<b>EID</b>	P	2	16	Environment identifier
<b>FADR</b>	P	3	32	Stored value of PC, used in exception handling
<b>SCR0-SCR3</b>	P	4 - 7	32	Supervisor-level scratch registers
<b>ESW</b>	P	8	32	Exception source word
<b>EMR</b>	P	9	32	Exception mask register
<b>ESR</b>	P	10	32	Exception set register
<b>ERR</b>	P	11	32	Exception reset register
<b>MADR</b>	P	12	32	Faulting memory address, used in exception handling
<b>TIMER</b>	P	13	32	Timer for programmable delay interrupts
<b>RCL</b>	P	14	32	Low-order bits of real-time clock
<b>RCH</b>	P	15	32	High-order bits of real-time clock
<b>NADR</b>	P	16	32	Stored value of PC after FADR, used in exception handling
<b>PC</b>	NA	NA	32	Program counter

**TABLE 1. Summary of registers**



**Figure 4: VC4aUY Processor Registers**

---

### 2.3. Operand Conventions

As stated earlier, memory operations are assumed to be aligned at 32-bit boundaries for the scalar integer and floating-point datapaths, and 256-bit boundaries for the wide datapath. Thus, on memory operations, the appropriate number of least significant bits in the address should be 0 (2 least significant bits for scalar integer and floating-point datapaths, least 5 significant bits for wideWord datapath).

Following the convention of the PowerPC, bits and bytes are stored in BigEndian order in memory.

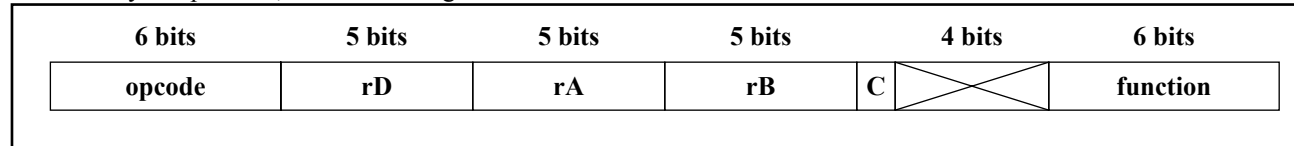
---

## Chapter 3 - ISA Summary

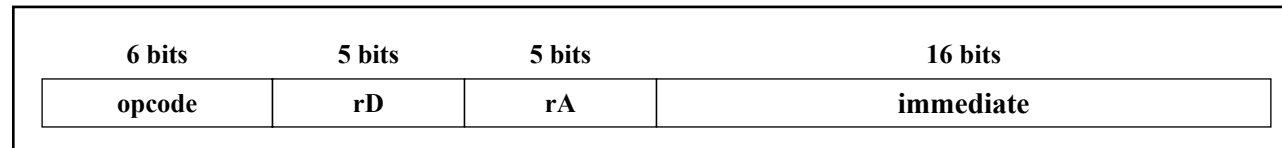
---

### 3.1. Scalar Instruction Formats

Details of the TA2\_SW instruction set architecture (ISA) can be found in the TA2\_SW RISC Processor Instruction Set Manual. An overview summary is provided in this section for reference. As shown in Figure 5, the TA2\_SW scalar instruction uses a three-operand format to specify two 32-bit source registers and a 32-bit target register. For arithmetic/logical instructions using this format, there is also a **C** bit to indicate whether the current instruction updates condition codes. However, the **C** bit indicates signed/unsigned arithmetic for multiply/divide instructions, since these instructions never update condition codes by definition. In lieu of a second source register, a 16-bit immediate value may be specified, as shown in Figure 6.

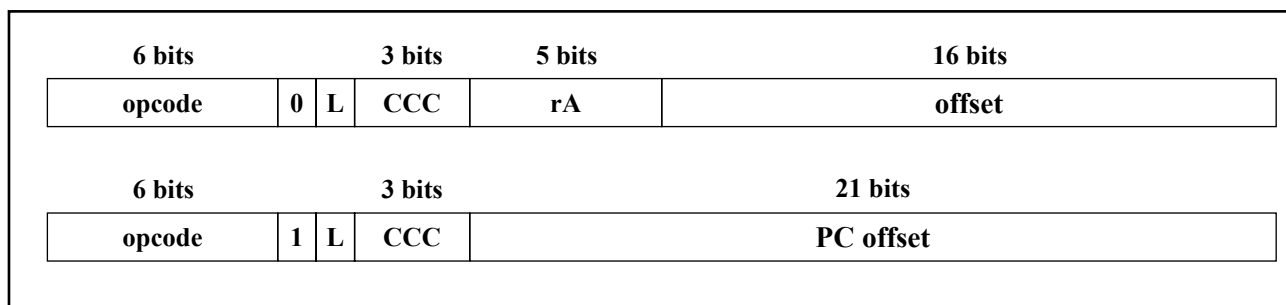


**Figure 5: Format R for Scalar (Integer and Floating-Point) Register Operations**



**Figure 6: Format I for Scalar Immediate Operations**

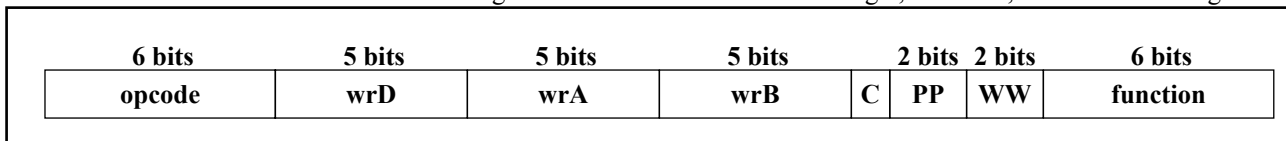
The branch instruction formats are shown in Figure 7. The branch target address may be PC-relative or calculated using a base register ORed with an offset. In both formats, the offset is in units of words, or 4 bytes, since instructions must be on a 4-byte boundary. Furthermore, the **L** bit specifies linkage, that is, whether a return instruction address should be saved in R31, referred to as a call instruction. Also, the **CCC** field specifies one of eight branch conditions: always, equal, not equal, less than, less than or equal, greater than, greater than or equal, or overflow. See the branch and call instruction descriptions in the TA2\_SW RISC Processor Instruction Set Manual for details.



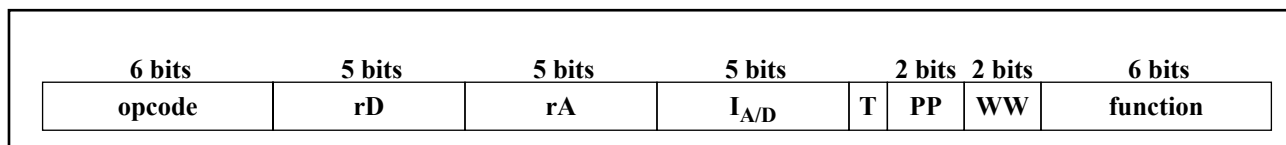
**Figure 7: Format B for Branches**

### 3.2. WideWord Instruction Formats

TA2\_SW WideWord operations are executed in a morphable arithmetic cluster which may be configured for WideWord operations. As shown in Figure 8, “WideWord Arithmetic/Logical Format,” WideWord instructions follow the general form of scalar instructions. Additional control information is included to manage the data fields of the WideWord and to modify the execution of the instruction. Figure 9 shows the format for transfers within the WideWord register file and across the scalar integer, scalar FP, and WideWord register files



**Figure 8: Format W for WideWord Arithmetic/Logical Operations**



**Figure 9: Format T for Wide-Word and Inter-Register File Transfers**

The control fields are defined as follows:

---

### ***WW (width)***

The ***WW*** field sets the width of the WideWord operands to eight, sixteen, or thirty-two bits, which primarily affects the shift operations and the configuration of the carry chain for additions and subtractions. For the merge instruction, these bits specify the condition on which the merge is based. The encoding of these bits is listed in the following table:

WW Value	Operand Width	Assembler Mnemonic
00	8 bits	b
01	16 bits	h
10	32 bits	w
11	Reserved	NA

### ***C (condition code enable)***

The ***C*** bit indicates whether condition codes will be updated as a result of the current instruction's execution. However, the ***C*** bit indicates signed/unsigned arithmetic for multiply, pack, and unpack instructions.

### ***PP (participation)***

The ***PP*** field interacts with condition codes to control whether a computation is performed on a given data field. The participation field can specify that a data field participate always, only if a condition local to its own data field is true, only if the data field is the leftmost field with a condition that is true, or only if the data field is the rightmost field with a condition that is true. The condition that is inspected for participation depends on the value of the ***PM*** (participation mode) register. Refer to Chapter 6 for more details. The encoding of the ***PP*** bits is listed in the following table:

PP Value	Participation Definition	Assembler Mnemonic
00	Always participate	a
01	Specified by local condition	o
10	Reserved	NA
11	Reserved	NA

### ***T (type)***

The ***T*** bit governs whether the current instruction operates on a vector or scalar. Depending on the function, ***rD*** or ***rA*** may specify a WideWord register. In this case, the ***T*** bit specifies whether the current transfer instruction refers to the WideWord register as a whole vector or instead uses ***I<sub>A/D</sub>*** to index a sub-field of the WideWord register.

### ***I<sub>A/D</sub>***

Value to be used as an index when a sub-field of a WideWord is involved in a transfer. Depending on the function, this index field may be an immediate or a scalar GPR specifier. Also, ***I<sub>A/D</sub>*** may be coupled with either ***rD*** or ***rA*** depending on the direction of the transfer as specified by the function.



### 3.3. Concise List

A concise list of the instructions in the TA2\_SW Instruction Set Architecture (ISA) is given in Table 2.

**TABLE 2. VC4aUY Instruction Set**

FUNC	DESCRIPTION	FUNC	DESCRIPTION	FUNC	DESCRIPTION
	<b>Scalar Instructions</b>		<b>WideWord Instructions</b>		<b>Branch Instructions</b>
<b>ADD</b>	Add	<b>WADD</b>	Add	<b>Bx</b>	Branch on scalar condition
<b>ADDE</b>	Add extended	<b>WADDE</b>	Add extended	<b>BAX</b>	Branch on all WideWord conditions
<b>ADDI</b>	Add immediate	<b>WSUB</b>	Subtract	<b>BNx</b>	Branch on no WideWord condition
<b>ADDIC</b>	Add immediate w/ condition codes	<b>WSUBE</b>	Subtract extended	<b>CALLx</b>	Call on scalar condition
<b>SUB</b>	Subtract	<b>WSUBU</b>	Subtract unsigned	<b>CALLAX</b>	Call on all WideWord conditions
<b>SUBE</b>	Subtract extended	<b>WMULES</b>	Multiply even signed	<b>CALLNx</b>	Call on no WideWord condition
<b>SUBU</b>	Subtract unsigned	<b>WMULEU</b>	Multiply even unsigned		<b>System Instructions</b>
<b>MUL</b>	Multiply	<b>WMULOS</b>	Multiply odd signed	<b>SYS</b>	System Call
<b>MULU</b>	Multiply unsigned	<b>WMULOU</b>	Multiply odd unsigned	<b>ICLI</b>	Instruction Cache Line Invalidate
<b>DIV</b>	Divide	<b>WAND</b>	And	<b>RFE</b>	Return from Exception
<b>DIVU</b>	Divide unsigned	<b>WNOT</b>	Bitwise inversion	<b>MTATR</b>	Move to address translation reg
<b>AND</b>	And	<b>WOR</b>	Or	<b>MFATR</b>	Move from address translation reg
<b>ANDI</b>	And immediate	<b>WXOR</b>	Xor	<b>MTPR</b>	Move to protected reg
<b>ANDIC</b>	And immediate w/ condition codes	<b>WSLL</b>	Shift left logical	<b>MFPR</b>	Move from protected reg
<b>NOT</b>	Bitwise inversion	<b>WSLLI</b>	Shift left logical immediate		
<b>OR</b>	Or	<b>WSRA</b>	Shift right arithmetic		<b>FPU Instructions</b>
<b>ORI</b>	Or immediate	<b>WSRAI</b>	Shift right arithmetic immediate	<b>FABS</b>	Floating-point absolute value
<b>ORIC</b>	Or immediate w/ condition codes	<b>WSRL</b>	Shift right logical	<b>FADD</b>	Floating-point add
<b>ORIS</b>	Or immediate shifted	<b>WSRLI</b>	Shift right logical immediate	<b>FDIV</b>	Floating-point divide
<b>XOR</b>	Xor	<b>WLD</b>	Load Reg from Mem	<b>FLD</b>	Floating-point load
<b>XORI</b>	Xor immediate	<b>WST</b>	Store Reg to Mem	<b>FMUL</b>	Floating-point multiply
<b>XORIC</b>	Xor immediate w/ condition codes	<b>WFABS</b>	Floating-point absolute value	<b>FNEG</b>	Floating-point negate
<b>SLL</b>	Shift left logical	<b>WFADD</b>	Floating-point add	<b>FST</b>	Floating-point store
<b>SLLI</b>	Shift left logical immediate	<b>WFMUL</b>	Floating-point multiply	<b>FSUB</b>	Floating-point subtract
<b>SRA</b>	Shift right arithmetic	<b>WFNEG</b>	Floating-point negate	<b>FTI</b>	Floating-point to integer conversion
<b>SRAI</b>	Shift right arithmetic immediate	<b>WFSUB</b>	Floating-point subtract	<b>ITF</b>	Integer to floating-point co version
<b>SRL</b>	Shift right logical	<b>WFTI</b>	Floating-point to integer conversion		
<b>SRLI</b>	Shift right logical immediate	<b>WITF</b>	Integer to floating-point co version		<b>Transfer Instructions</b>
<b>LD</b>	Load Reg from load buffer if possible	<b>WPRM</b>	Permute	<b>MVFF</b>	Move FPU to FPU
<b>ST</b>	Store Reg to store buffer if possible	<b>WPRMI</b>	Permute immediate	<b>MVFS</b>	Move FPU to scalar
<b>LDBI</b>	Load buffer invalidate	<b>WMRG</b>	Merge based on condition codes	<b>MVFW</b>	Move FPU to WW
<b>STBF</b>	Store buffer flush	<b>WPKS</b>	Pack using signed arithmetic	<b>MVFWI</b>	Move FPU to WW, indirect
		<b>WPKU</b>	Pack using unsigned arithmetic	<b>MVSF</b>	Move scalar to FPU
	<b>Miscellaneous Instructions</b>	<b>WUPKL</b>	Unpack low-order byte/halfword	<b>MVSW</b>	Move scalar to WW
<b>MTSPR</b>	Move to special-purpose reg			<b>MVSWI</b>	Move scalar to WW, indirect
<b>MFSPR</b>	Move from special-purpose reg			<b>MVWF</b>	Move WW to FPU
<b>LOKL</b>	Lock Load			<b>MVWFI</b>	Move WW to FPU, indirect
<b>LOKS</b>	Lock Store			<b>MVWS</b>	Move WW to scalar
<b>PROBE</b>	Probe address to determine locality			<b>MVWSI</b>	Move WW to scalar, indirect
<b>ELO</b>	Encode leftmost one	<b>TKLD</b>	Token Load	<b>MVWW</b>	Move WW to WW
<b>CLO</b>	Clear leftmost one	<b>TKST</b>	Token Store	<b>MVWWI</b>	Move WW to WW, indirect

---

## Chapter 4 - Execution Pipeline and Scalar Datapath

---

### 4.1. Introduction

TheTA2\_SW execution pipeline is a five-stage unit that is used to control the operation of the scalar, floating-point and WideWord datapaths. Because the combined pipeline and scalar datapath are quite similar to familiar RISC processor architectures, the operation of these units are detailed together to simplify description. The stages of the pipeline are named here, with an explanation of the major events occurring within that stage of execution.

#### 4.1.1. Pipeline Stages

We establish the convention that each stage views its local instruction and output to be synchronized at the next clock edge as the *current* instruction. While from an external view, there are *five* instructions “currently” executing, the ALU stage sees an opcode, two operands, and control and stored state as components of the “current” instruction. This view of execution local to each stage is the convention used in all descriptions of the pipeline.

##### ***F - instruction fetch***

The F stage of the pipeline is where the address of the current instruction is applied to the instruction cache and the instruction is located. At the end of the cycle the output of the instruction cache is latched into the first register stage of the pipeline.

*During the F stage, the address for the next instruction is calculated. Note that the calculation applies to sequential addresses as well as branches.*

##### ***D - register decode***

During the D stage, operands for the current instruction are selected from the register file or the most recent value in the pipeline forwarding logic. In the case of an immediate instruction, the immediate field of the current instruction is routed to the SRC2 pipeline. The result is latched into the datapath D-stage registers.

##### ***X - execute***

Depending on the instruction, the X stage performs the computation defined by the opcode. For memory operations, the effective address is calculated in the X stage.

##### ***M - memory***

Register load and store instructions require memory accesses. To maintain consistency with the normal register-write logic, memory operations are begun during the M cycle, and the pipeline is stalled until memory arbitration and the required read operation has been performed. During memory write operations, the pipeline is released as soon as arbitration grants access to the memory.

##### ***W - write***

During the W stage, the register file is written with the result of the current operation, whether a computation or a memory read.

### 4.2. Major Signal Paths

Major data and control paths of theTA2\_SW RISC processor are shown in Figure 10 and Figure 11. Execution pipeline logic is depicted in the shaded area of the figures, while the unshaded area of the figures shows the control pipeline and scalar datapath.

#### 4.2.1. Scalar Data Path

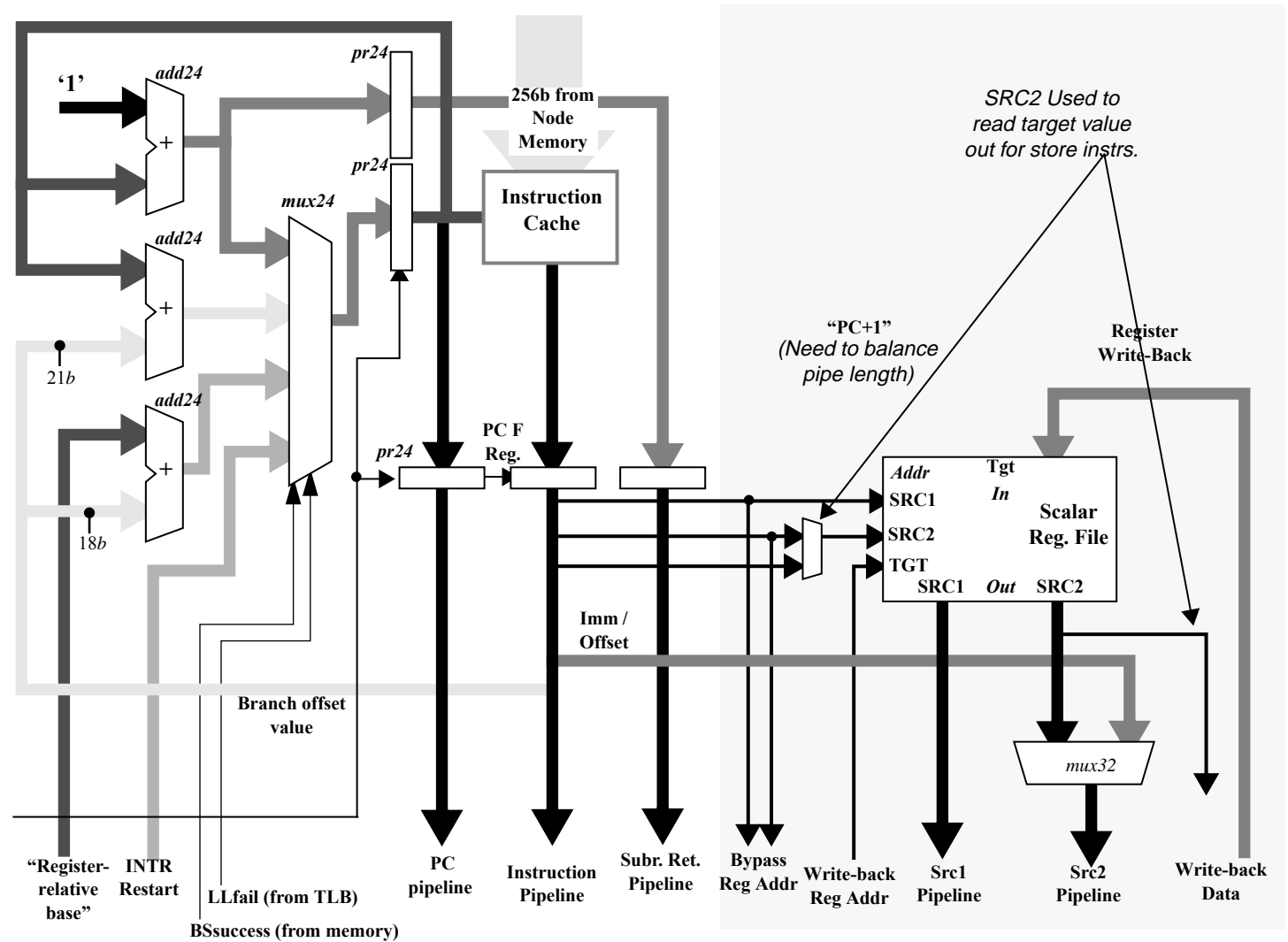


Figure 10: 5-Stage Execution Pipeline (F & D Stages)

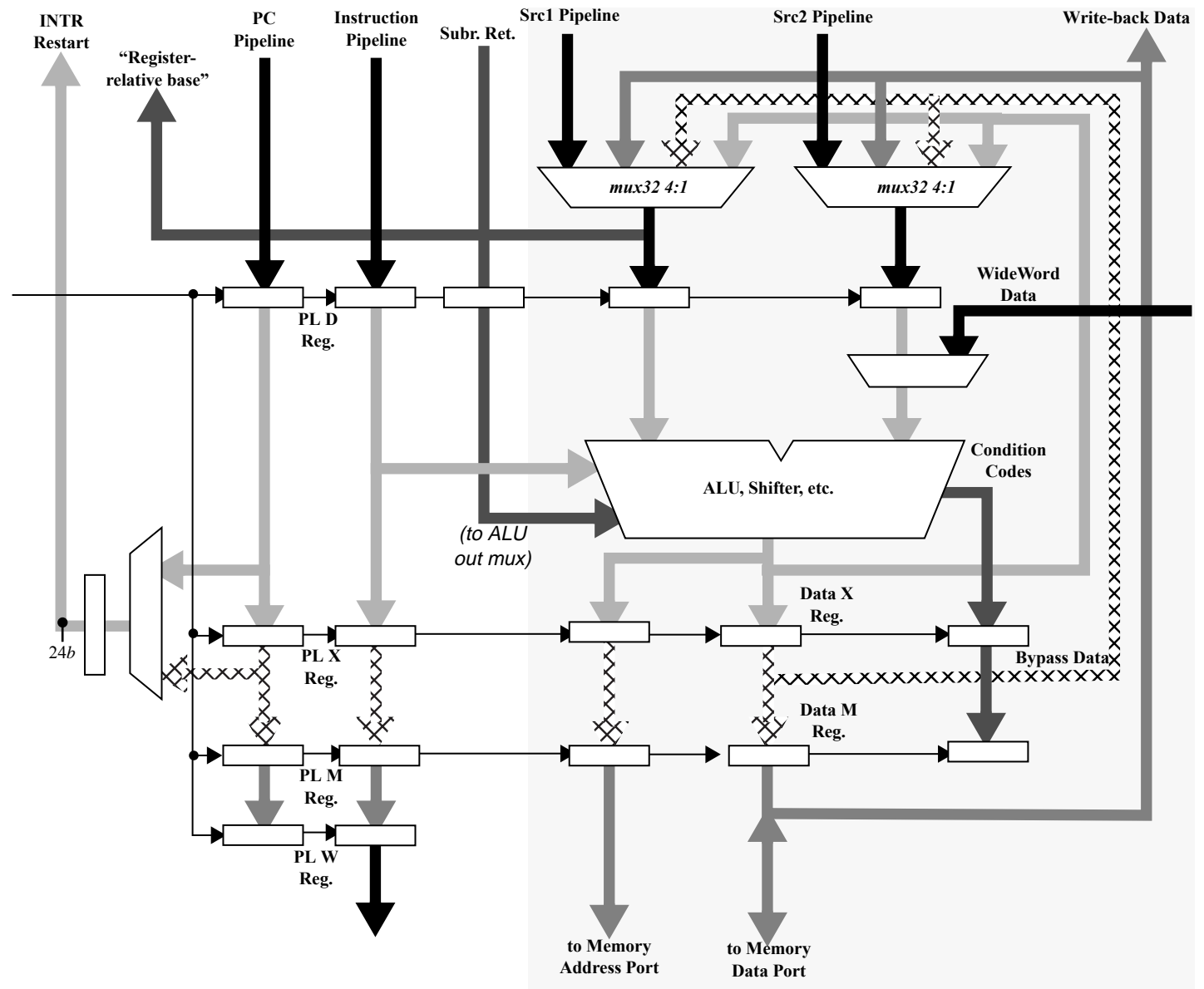


Figure 11: 5-Stage Execution Pipeline (D through W Stages)

---

### 4.3. Scalar Computing Functions

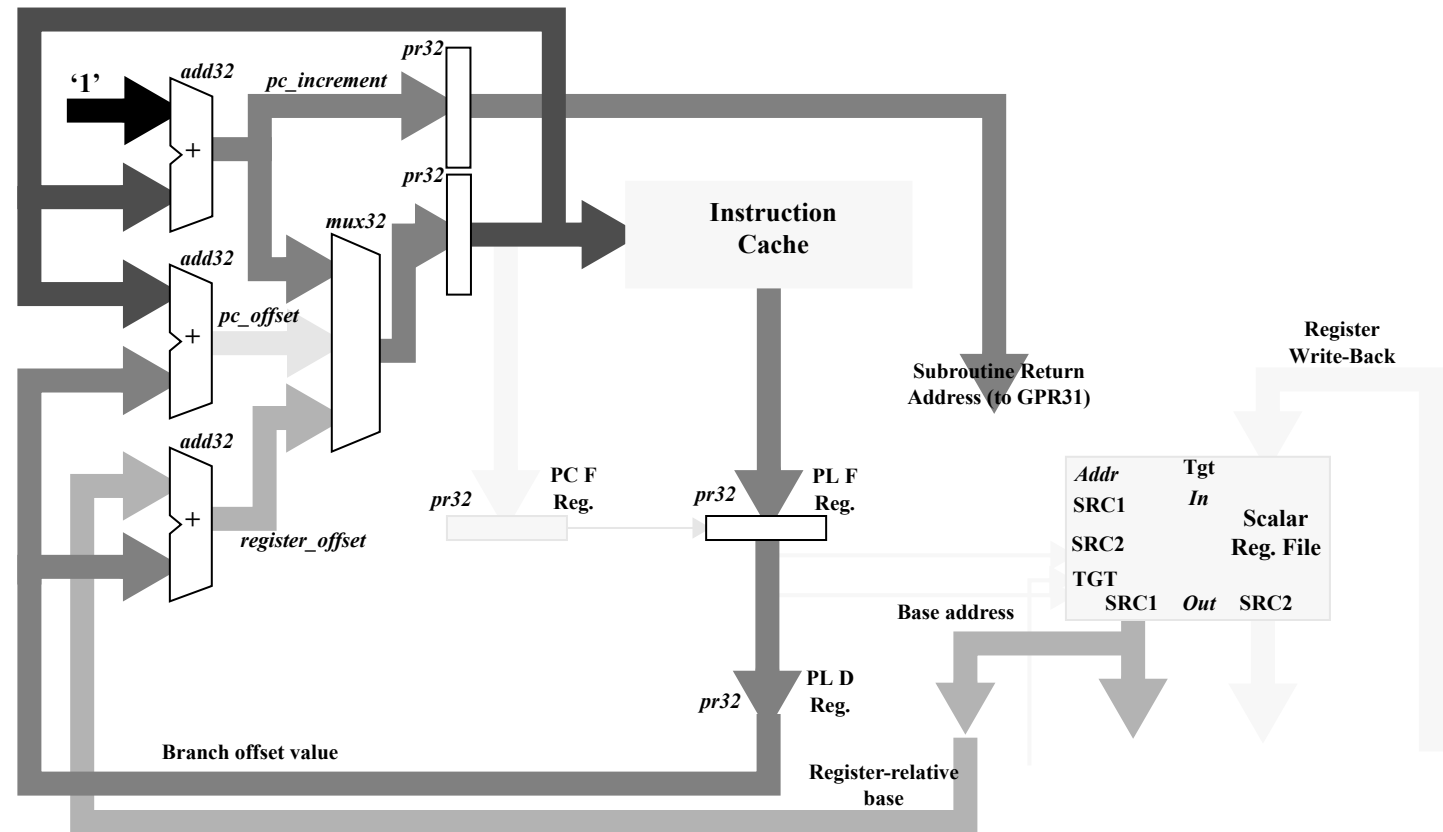
The scalar datapath performs operations on objects of 32 bits or less. Refer to the TA2\_SW Instruction Set Architecture document for a complete description of these operations.

### 4.4. Pipeline Analysis

Numerous examples of a five-stage pipeline exist in the literature, providing a starting design-point for new machines, including TA2\_SW. We perform an analysis of the TA2\_SW pipe to ensure no undue overhead is incurred by branches or other changes in program flow.

#### 4.4.1. Address Calculations

Figure 12 below is excerpted from the earlier execution pipeline illustration, Figure 10. The address calculation portion of the pipeline has been highlighted to clarify the several parallel paths used to develop the address of the next instruction to be executed. Address computations are performed in parallel to guarantee the fastest possible operations. The address calculations indicated in the figure are: *pc\_increment*, *pc\_offset*, and *register\_offset*, which correspond to the types of branches supported by TA2\_SW.



**Figure 12: Instruction-Address Pipeline**

#### 4.4.2. Branch Pipeline States

As shown in Figure 12, a branch/call instruction incurs one delay slot because the branch cannot be resolved until the decode stage of the pipeline. This delay slot is exposed to the compiler to create the opportunity for the compiler to reschedule instructions and exploit the clock cycle for the delay slot. In the event a code sequence cannot be rescheduled, a NOP instruction inserted after the branch/call instruction is needed to ensure proper operation.

### 4.5. Pipeline Hazards

In pipelined systems, hazards occur when an operation is begun before another has completed, or before required results are available. In TA2\_SW, these are broken down into three classes: *instruction sequences*, *register operations*, and *memory operations*. Each of these hazard classes is described below.

#### 4.5.1. Instruction Sequences

There are several instances of instructions that incur hazards due to “extra” time required for completion. Among these instructions are integer multiply and divide. When these instructions reach the execute stage of the pipeline, they are forked off to a separate compute unit which

---

writes its results to the special-purpose HI and LO registers when the computation is completed. The execution pipeline continues processing concurrently. Thus, software scheduling is necessary to ensure the contents of HI and LO are read only after such instructions have completed. Refer to the Instruction Set Manual for more details on such instructions.

#### 4.5.2. Register Operations

Register hazards occur when an instruction requires an operand that is currently in the data pipeline. In the simplest case, consider a stream of instructions where a register is required in the same clock cycle where it is being written into the register file. This hazard can be very simply eliminated by requiring register writes to complete in the first half of each clock cycle, and performing all register reads during the second half. This is well within the capabilities of the technology.

Consider the following code sequence, where an operand is not ready:

```
ADD    R3, R1, R2           /* R3 = R1 + R2 */
ADD    R5, R3, R4           /* R5 = R3 + R4 */
```

Because R3 is emerging from the ALU as the first instruction finishes execution, it is not available to be fetched from the register file. This hazard requires *bypassing* or *forwarding* to get the most recent copy of a register from a later stage in the pipeline, and move it to the ALU inputs. Selection is performed by comparing the destination address of every register in the pipeline against the register specification accessing the register file. The most recent copy (closest to the ALU) is selected, resolving events where several copies of a register are in the pipeline.

#### 4.5.3. Memory Operations

Memory-related hazards can occur in TA2\_SW. These are caused by the proximity of register load and store instructions. Consider the following code sequence, which is typical of moving data for further processing:

```
MOV    R1, R0               /* initialize the index */
LD     R2, TABL1, R1         /* */
ST     R2, TABL2, R1         /* */
ADD    R1, 0x1
```

Now it is impossible for both the execution pipeline and the memory to respond to these two instructions as written. First, the pipeline can't store a value that has not yet loaded: the register write-back stage is *after* the memory write stage. Second, there is no guarantee that the objects TABL1 and TABL2 are located in the same open row in memory. As a result, an unknown number of delays will occur before the store request will start in the memory.

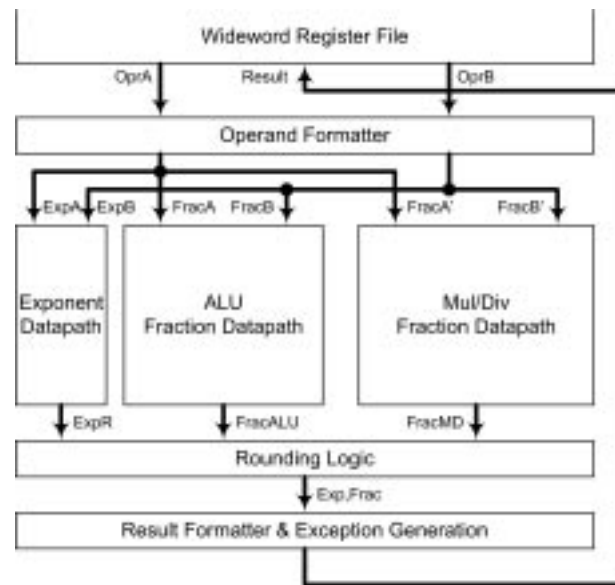
---

## Chapter 5 - Floating-Point Datapath

---

### 5.1. FPU Microarchitecture

TheTA2\_SW scalar FPU implements a subset of the IEEE-754 floating-point standard. Since target applications are mostly from the embedded signal processing realm, only single-precision numbers are supported. To achieve a better area-performance solution, operations on denormalized numbers are not supported and cause exceptions. In addition, whenever a result is a denormalized number, an underflow exception is raised and the minimum normalized number is produced for output. The inexact exception flag on division operations is not IEEE-754 compliant, which is common for multiplicative division algorithms. Additional operations are necessary to correct this. Other exception flags – Invalid, Divide by Zero, Overflow, Underflow and Inexact (except divide) – are accurately generated as specified by the IEEE-754 standard. TheTA2\_SW FPU implements only the “round to nearest” rounding mode. Figure 13 depicts the microarchitecture



**Figure 13: TA2\_SW FPU microarchitecture**

of the FPU. The FPU has two main blocks: ALU and Mul/Div. Exponent computation functions for both blocks are combined in one datapath to reduce area. Similarly, converting logic to/from the internal number format and rounding logic are shared for both of the datapaths. As only one instruction can be issued at each cycle, combining common datapaths does not suffer any performance penalty. Input registers for the ALU and the Mul/Div blocks are controlled by separate enable signals so that only one of the datapaths is active for each instruction. Table 3 shows the supported floating-point instructions and their pipeline latency and throughput.



---

**TABLE 3. FPU instruction latency/throughput**

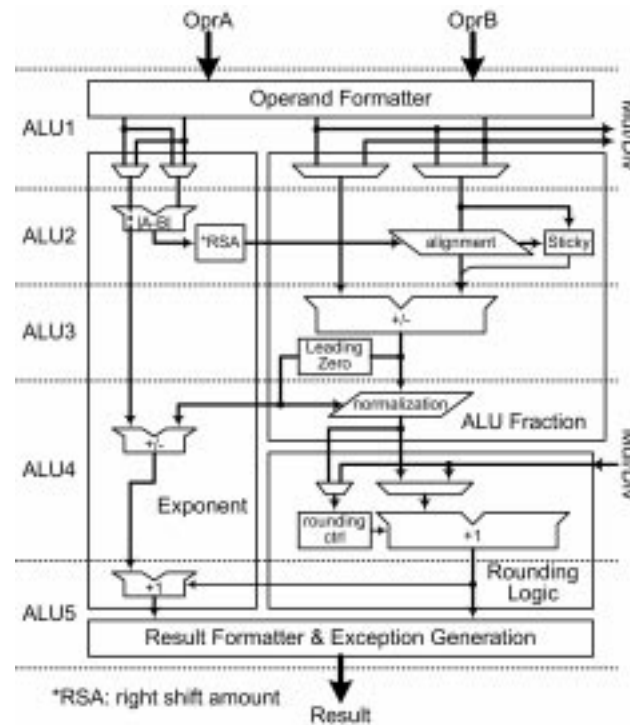
Instruction	Latency	Throughput
Add/Subtract	5	1
FP2Int/Int2FP	5	1
Absolute/Negate	5	1
Multiply	5	1
Divide	12	5/8*

\* 5 cycles for consecutive divide instruction and 8 cycles for other subsequent instruction

## 5.2. FPU ALU

A block diagram of the ALU is shown in Figure 14. Add/Sub instructions proceed by swapping operands if necessary, aligning the fraction of the smaller operands, computing the fraction, normalizing the fraction with adjustment of the exponent, rounding and generating the exception flag, if any. The exponent datapath includes three exponent adders that are also used for multiply and divide instructions. For Absolute/Negate instructions, OprB is preset to zero by the operand formatter then added to OprA in both the fraction and exponent datapaths. The controller determines the sign bit of the result based on the sign bit of OprA. For the Fp2Int (Floating-point to integer) instruction, the fraction is shifted right depending on the value of exponent ( $157 - \text{ExpA}$ ), forming a 31-bit unsigned integer. If the floating-point number is negative, the fraction is inverted for the two's complement representation. Rounding and overflow detection is carried out thereafter. For Int2Fp (Integer to floating-point instruction, the fraction is first converted to sign-magnitude format by conditionally inverting if the sign is

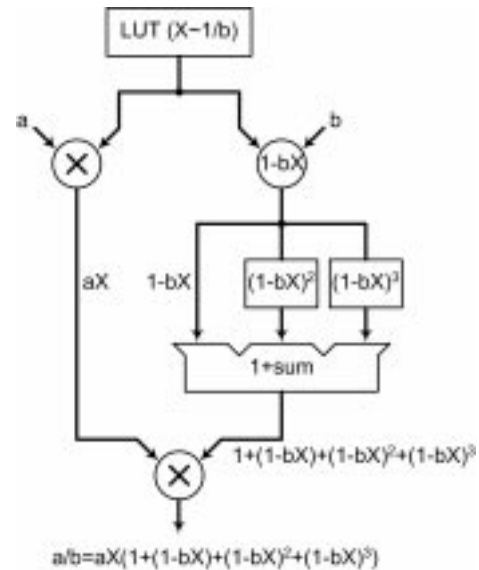
negative. Then the result is shifted left to remove leading zeros. The exponent is adjusted accordingly by this leading zero value. Note that the exponent value of OprA is preset to 157, which corresponds to  $2^{30}$  in integer form.



**Figure 14: TA2\_SW FPU ALU datapath**

### 5.3. FPU Multiplier/Divider Unit

To meet performance requirements of modern scientific applications such as 3D graphics rendering, high performance is crucial for division as well as multiplication. High-radix SRT dividers based on the digit recurrence algorithm are widely used for modern microprocessors. However, this type of divider is extremely area-intensive and not necessarily the appropriate design for embedded processors. Since the TA2\_SW chip architecture includes many components, a good area-performance solution is the primary design goal. To achieve this, we adapted the multiplicative division algorithm proposed by Liddicoat and Flynn, which is based on Taylor series expansion, as shown in Figure 15. This algorithm achieves fast computation by using parallel squaring and cubing units, which compute the higher-order terms significantly faster than the traditional serial multipliers with a relatively small hardware overhead. There are three major multiply operations to produce a quotient with 0.5 ulp (unit in the last place) error.



**Figure 15: Liddicoat and Flynn division algorithm**

One additional multiply operation is required for exact rounding. To maximize the area efficiency, all of these multiply operations are executed by one multiplier. By sharing the multiplier, the pipeline latency increases by four times. However, through careful pipeline scheduling, we were able to achieve high throughput for consecutive divide instructions. A lookup table for an initial seed value is implemented using a 128x7-bit ROM. A two-stage pipelined multiplier is used for better synthesis results under the high-performance timing specifications.

tions. Figure 16 shows the block diagram of the fused multiplier/divider unit, and Table 4 summarizes the operations in each cycle for the divide instruction.

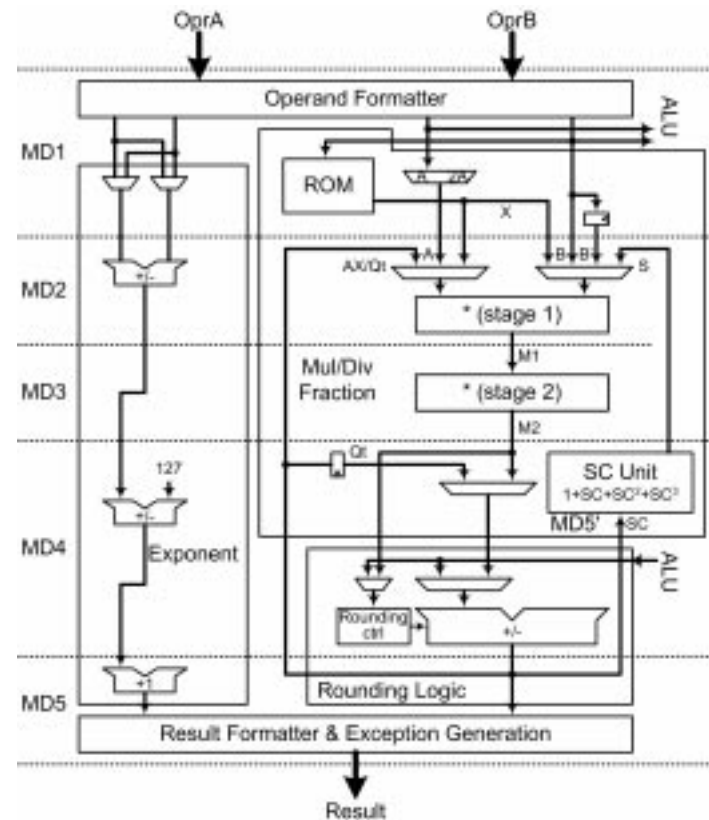


Figure 16: TA2\_SW FPU multiplier/divider datapath

---

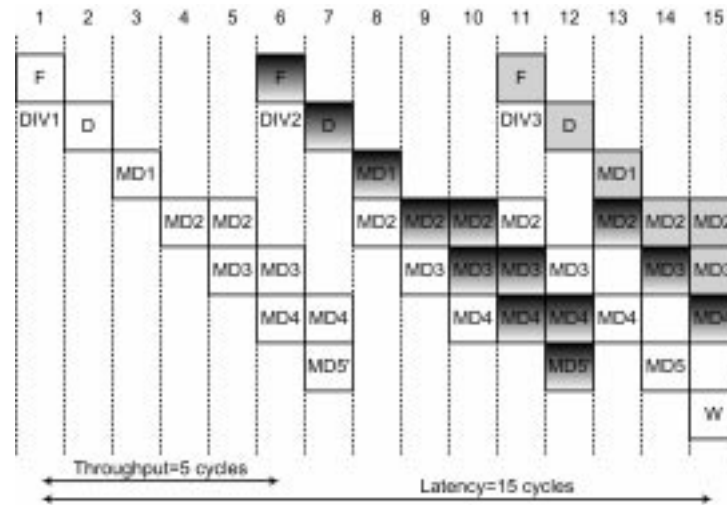
**TABLE 4. Steps for divide operation**

Steps	Operation	Pipeline Stage
1	$X = \text{ROM}(b)$	MD1
2	$M1 = b * X$ (stage1)	MD2
3	$M2 = b * X$ (stage2), $M1 = a * X$ (stage1)	MD3/MD2
4	$SC = 1 - M2$ , $M2 = a * X$ (stage2)	MD4/MD3
5	$S = 1 + SC + SC^2 + SC^3$ , $AX = M2$	MD5*/MD4
6	$M1 = AX * S$ (stage1)	MD2
7	$M2 = AX * S$ (stage2)	MD3
8	$Qt = \text{trunc}(M2) + 1$	MD4
9	$M1 = b * Qt$ (stage1)	MD2
10	$M2 = b * Qt$ (stage2)	MD3
11	$R = \text{round}(Qt)$	MD4
12	$\text{Result} = \text{format}(R)$	MD5

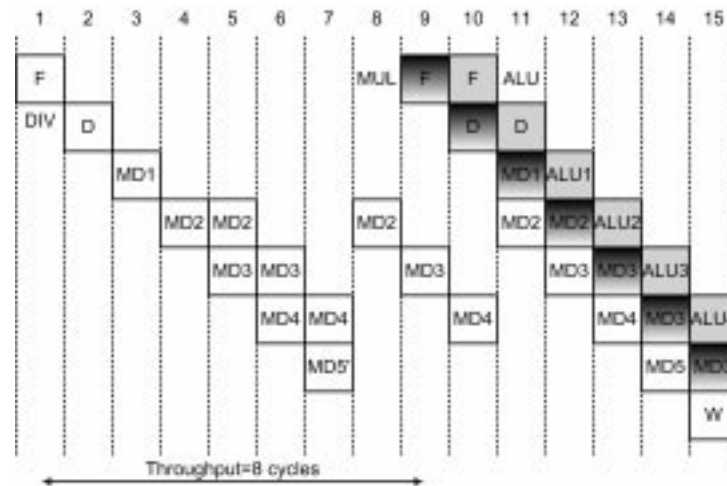
#### 5.4. FPU Pipelining

Figure 17 shows the pipeline diagram for three consecutive divide instructions. Although 12 clock cycles are required to complete one divide instruction, the pipeline is designed such that divide instructions can be issued every five clock cycles. If any other type of instruction follows

a divide instruction, a pipeline stall for seven clock cycles is required to ensure in-order completion as shown in Figure 18. All other combinations of instructions run without pipeline stalls.



**Figure 17: Pipeline timing for consecutive divide instructions**



**Figure 18: Pipeline timing for divide instruction followed by other type of instruction**

There is also an FPU in the FPCA portion of TA2\_SW that can be used in either streaming or threaded WideWord mode. This FPU does not support divide operations, and as a result, is a 3-stage pipelined FPU. The principles of operation of this FPU are similar to the scalar FPU described previously in this section.

---

## Chapter 6 - WideWord Datapath

---

### 6.1. WideWord Features

As noted in earlier chapters, theTA2\_SW RISC processor has the ability to control an external arithmetic cluster as a wide datapath. When controlled in this manner the wide datapath supports a number of interesting features. First, it supports *selective execution* of instructions on sub-fields within a 256-bit value. Under selective execution, only the results corresponding to the data paths that participate in the computation are written back, or committed, to the instruction's destination registers. The data field that participate in the conditional execution of a given instruction are derived from the condition codes or the mask register, plus the instruction's *participation field*. The conditions used (condition codes or mask register) are specified in the *participation mode* register. The instruction's participation field determines how the condition code (or mask register) bits are combined to specify the participation of each data path.

#### 6.1.1. Participation field

Each WideWord instruction with support for conditional execution has a 2-bit participation field. The participation field specifies two ways in which the condition code (or mask register) bits are combined for determining participation of each data path: (1) *Always participate*, where all data field participate; (2) *Local participation*, where a data field participates only if a condition local to its own data path is true. The encoding of the participation field (*PP*) bits is described in the documentTA2\_SW RISC Processor Instruction Set Manual, and is also listed in the following table:

PP Value	Participation Definition
00	Always participate
01	Local participation
10	Reserved
11	Reserved

#### 6.1.2. Participation Mode

The conditions that are inspected for participation depend on the value of the Participation Mode (*PM*) register. The PM register is a 5-bit register that is read/written using the `mfspr/mtspr` instructions. The conditions correspond to the condition codes EQ, GT, LT, OV or the mask register M. The encoding of the Participation Mode is shown in the following table:

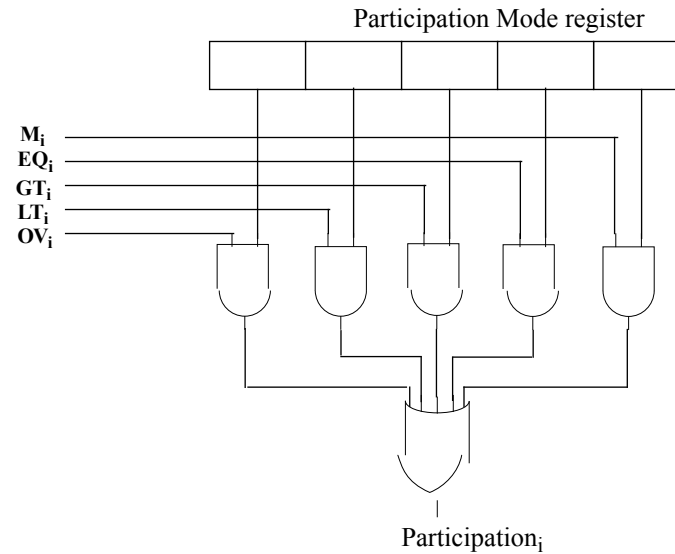
PM Value	Mask/Condition Code
00001	M
00010	EQ
00100	GT
01000	LT
10000	OV

Any combination of the 5 conditions listed in the table can be used to determine participation. For instance, if the PM value is 00110, the EQ and GT condition codes are ORed together to determine participation.

In addition, if the mask register is updated, the participation mode register is automatically updated to select M for participation.



The figure below illustrates an implementation of local participation for data path  $i$  (note that this simple example is not a complete implementation of a participation bit and does not include the participation field bits)



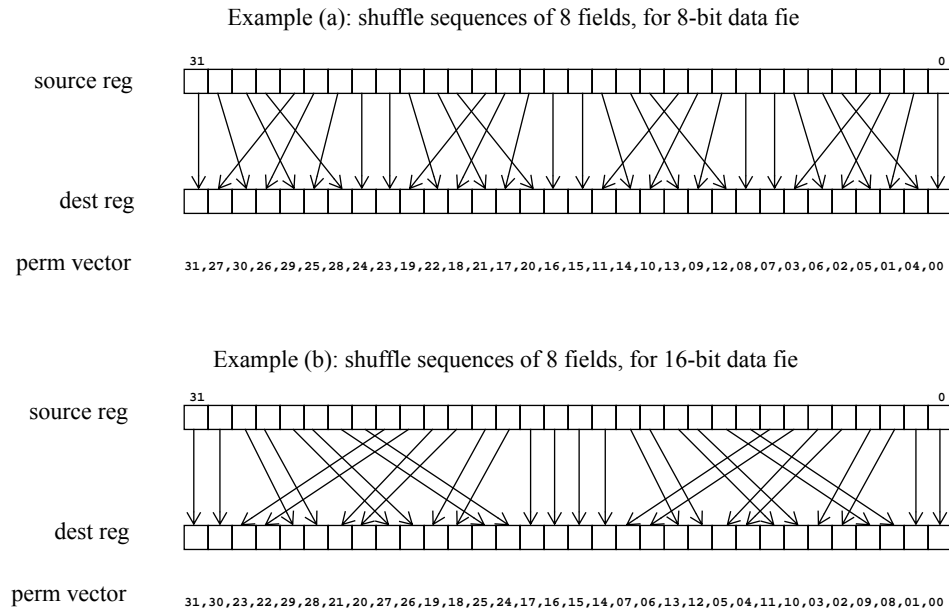
**Figure 19: Example of participation bit derived from PM register and condition codes**

### 6.1.3. Setting the condition bits for participation

For simplicity, the WideWord ALU performs conditional write-backs (commits the results) on 8-bit datapaths, independently of the datapath width of the instruction. Conditional operations on 16-bit or 32-bit data paths assume that the condition bits for participation (condition codes or mask register) are set consistently with the current datapath width. For example, an instruction that operates on 32-bit data fields should have a 32-bit result written back to the destination register, for each participating 32-bit data field. Therefore, since the WideWord ALU performs conditional write-backs of 8-bit values, the 4 consecutive bits of the condition code/mask register corresponding to a 32-bit datapath should be set consistently (either all ones, for participation, or all zeros). It is the programmer's responsibility to ensure that the conditions for participation are consistent with the datapath width, either by setting the mask register or by performing a previous operation with the same datapath width to set the condition codes.

## 6.2. Permutation

The WideWord permutation network supports fast alignment and reorganization of data in wide registers. The permutation network supports general permutations of 8-bit data fields that is, any 8-bit data field of the source register can be moved into any 8-bit data field of the destination register. A permutation is specified by a *permutation vector*, which is a 256-bit object containing 32 indices corresponding to the 32 8-bit data fields of a WideWord. Each 8-bit field of a permutation vector corresponds to the same 8-bit data field of the destination register, and contains the index of the source data field to be moved into that destination field. The figure below illustrates a permutation on 8-bit and 16-bit data paths, and the corresponding permutation vectors.



**Figure 20: Example of permutation vectors for 8-bit and 16-bit data paths**

Two types of permutation operations are supported: `wprm` and `wprmi`. In `wprm`, the permutation vector is contained in a general-purpose wide register, allowing permutation vectors to be loaded from memory and manipulated using WideWord operations. `wprmi` selects a permutation vector from a lookup table, supporting faster permutations (one operation) for the set of frequently used permutation vectors in the table. The hardwired permutation vectors are listed in the following table, and the permute instructions are described in more detail in the document TA2\_SW RISC Processor Instruction Set Manual.

index	vector
0x00	0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
0x01	0x0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00
0x02	0x02030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001
0x03	0x030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102
0x04	0x0405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203
0x05	0x05060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001020304
0x06	0x060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405

index	vector
0x07	0x0708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203040506
0x08	0x08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001020304050607
0x09	0x090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708
0x0A	0x0A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203040506070809
0x0B	0x0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A
0x0C	0x0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B
0x0D	0x0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C
0x0E	0x0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D
0x0F	0x0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E
0x10	0x101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F
0x11	0x1112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10
0x12	0x12131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011
0x13	0x131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112
0x14	0x1415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213
0x15	0x15161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011121314
0x16	0x161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415
0x17	0x1718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213141516
0x18	0x18191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011121314151617
0x19	0x191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718
0x1A	0x1A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213141516171819
0x1B	0x1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A
0x1C	0x1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B
0x1D	0x1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C
0x1E	0x1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D
0x1F	0x1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E
0x20	0x00020406080A0C0E10121416181A1C1E101030507090B0D0F11131517191B1D1F
0x21	0x010003020504070609080B0A0D0C0F0E111013121514171619181B1A1D1C1F1E
0x22	0x03020100070605040B0A09080F0E0D0C13121110171615141B1A19181F1E1D1C
0x23	0x07060504030201000F0E0D0C0B0A090817161514131211101F1E1D1C1B1A1918
0x24	0x0F0E0D0C0B0A090807060504030201001F1E1D1C1B1A19181716151413121110
0x25	0x1F1E1D1C1B1A191817161514131211100F0E0D0C0B0A09080706050403020100
0x26	0x0002010304060507080A090B0C0E0D0F1012111314161517181A191B1C1E1D1F
0x27	0x0004010502060307080C090D0A0E0B0F1014111512161317181C191D1A1E1B1F

index	vector
0x28	0x00080109020A030B040C050D060E070F10181119121A131B141C151D161E171F
0x29	0x0001040508090C0D1011141518191C1D020306070A0B0E0F121316171A1B1E1F
0x2A	0x02030001060704050A0B08090E0F0C0D12131011161714151A1B18191E1F1C1D
0x2B	0x06070405020300010E0F0C0D0A0B080916171415121310111E1F1C1D1A1B1819
0x2C	0x0E0F0C0D0A0B080906070405020300011E1F1C1D1A1B18191617141512131011
0x2D	0x1E1F1C1D1A1B181916171415121310110E0F0C0D0A0B08090607040502030001
0x2E	0x000104050203060708090C0D0A0B0E0F101114151213161718191C1D1A1B1E1F
0x2F	0x0001080902030A0B04050C0D06070E0F1011181912131A1B14151C1D16171E1F
0x30	0x0001020308090A0B1011121318191A1B040506070C0D0E0F141516171C1D1E1F
0x31	0x04050607000102030C0D0E0F08090A0B14151617101112131C1D1E1F18191A1B
0x32	0x0C0D0E0F08090A0B04050607000102031C1D1E1F18191A1B1415161710111213
0x33	0x1C1D1E1F18191A1B14151617101112130C0D0E0F08090A0B0405060700010203
0x34	0x0001020308090A0B040506070C0D0E0F1011121318191A1B141516171C1D1E1F
0x35	0x0001020310111213040506071415161708090A0B18191A1B0C0D0E0F1C1D1E1F
0x36	0x1011121300010203141516170405060718191A1B08090A0B1C1D1E1F0C0D0E0F
0x37	0x08090A0B0C0D0E0F000102030405060718191A1B1C1D1E1F1011121314151617

### 6.3. Merge

The WideWord unit supports a special instruction (`wmrg`) for merging data from two source registers according to a given condition. The condition is specified by the WW field of the instruction, and can be one of the condition codes EQ, LT or GT, or the M register. The following table shows the encoding of the WW field

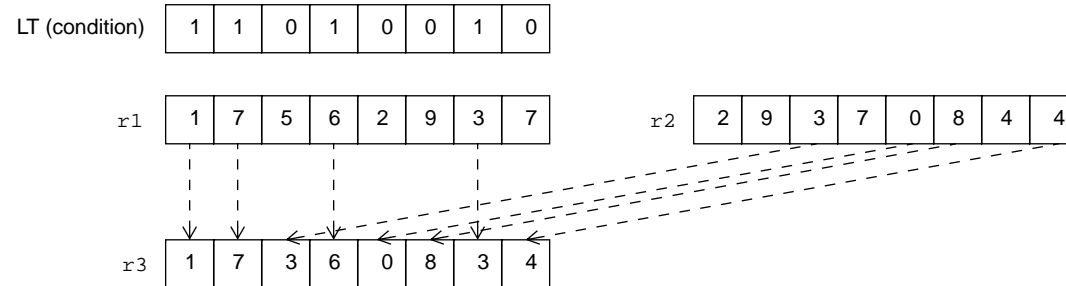
WW Value	CC
00	EQ
01	LT
10	GT
11	M

The figure below illustrates a merge operation using the condition LT. The condition codes are set by a previous `wsubc` instruction with the same data path width as the `wmrg` instruction.

```

wsubcw r4, r1, r2
wmrgltw r3, r1, r2

```



## 6.4. Transfers

A set of *transfer instructions* allows data to be moved between WideWord and other register files (1) between wide registers and scalar integer registers; (2) from wide register to wide register; and (3) between wide registers and scalar floating-point registers. The transfer functions where the source is a scalar value (scalar integer or floating-point register or a data field in a wide register), and the destination is a wide register allow the source data to be replicated and stored into all the fields of the destination

The complete set of transfer instructions is described in detail in the TA2\_SW RISC Processor Instruction Set Manual.

---

## Chapter 7 - Instruction Cache

---

### 7.1. Introduction

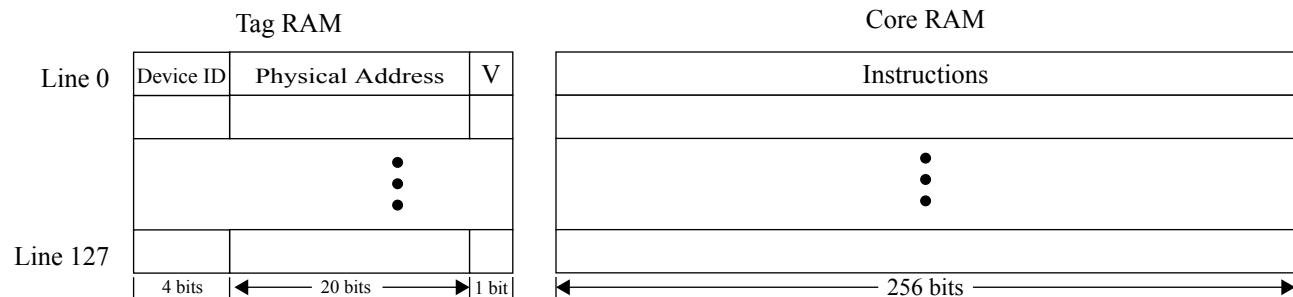
It is of critical importance to keep instruction fetches from interfering with the flow of operand data from the node memories. In addition to the reduction of operand data bandwidth due simply to contention, instruction fetches from memory reduce bandwidth even further due to the resulting increase in memory latency because they disrupt reference locality. Since the code segment of an application is placed in a different area of memory from the data segment, interleaving instruction fetches with operand fetches from memory would effectively randomize memory accesses that could have otherwise been satisfied in a page-mode fashion. TA2\_SW avoids most of the bandwidth losses by implementing a small instruction cache.

### 7.2. Instruction Cache Description

The TA2\_SW RISC processor contains a 4-Kbyte, direct-mapped instruction cache. The cache line size is 32 bytes, and each cache line can be loaded or invalidated individually. In addition, the entire cache can be invalidated by disabling the cache. The TA2\_SW architecture does not support self-modifying code, so the instruction cache has no write-back capability. The cache does not contain a snooping port and is therefore not kept coherent with memory automatically. Kernel software is responsible for invalidating stale cache lines when the backing memory for those lines is being loaded with new code.

### 7.3. Instruction Cache Organization

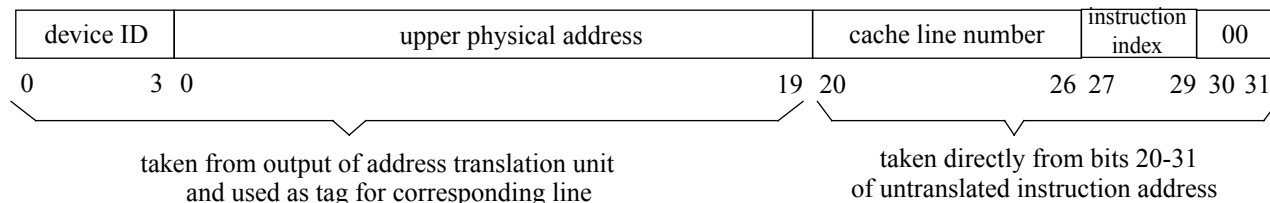
The cache consists of three major components: core ram, tag ram, and the controller. A diagram showing the organization of the core ram and tag ram is shown in Figure 21. The core RAM consists of 128 lines, where each line is 256 bits long. Each line is then capable of storing eight 32-bit instructions. The tag RAM contains a 24-bit tag, 4 bits of device ID and 20 bits of physical address, for each line of core RAM, although the physical address field could be reduced to match the amount of physical memory actually present and thereby optimize the storage and performance of tag accesses. Each tag RAM line also contains a valid-bit to indicate whether the line contents is empty or it contains valid information.



**Figure 21: Instruction Cache Organization**

An instruction virtual address generated by the processor instruction fetch engine is translated/decoded as shown in Figure 22 for determining placement or validity within the cache. The instruction cache unit operates in conjunction with the address translation unit. For example, the least significant 12 bits of the 32-bit instruction virtual addresses generated by the processor instruction fetch engine are specified to be unaffected by the address translation process. Therefore, the seven most significant of these 12 bits, which correspond to bits 20 through 26 of a TA2\_SW node bus address, can be safely used to index into the cache simultaneously with the translation of the upper 20 bits. By the

time the appropriate tag has been accessed, the translation has taken place, so that the tag contents can be compared with the physical address, including device ID. The translated upper 20 bits, corresponding to bits 0 through 19 of a TA2\_SW node bus address, and 4-bit device ID are then used as the tag information for a cache line. Bits 27 through 29 of a processor instruction virtual address are used to select



**Figure 22: Use of Virtual/Physical Instruction Address Bits in Instruction Cache Operation**

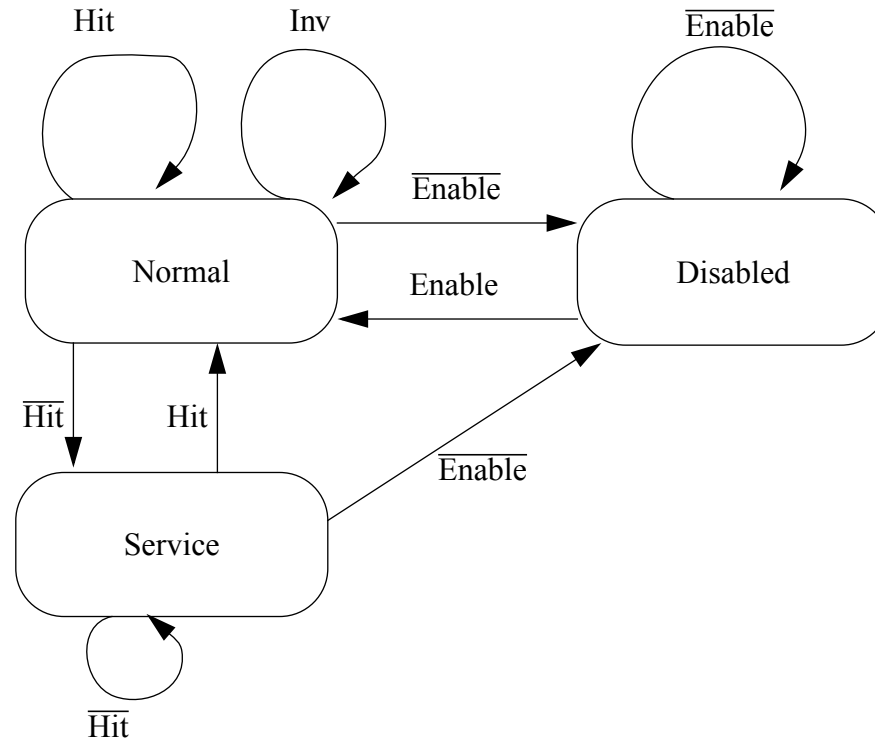
a specific instruction within a cache line. Refer to Chapter 8 on address translation for more information on how virtual addresses are converted to TA2\_SW node bus physical addresses and device IDs.

## 7.4. Instruction Cache Operation

The operation of the instruction cache is best described by defining the tasks of the cache controller. The controller is responsible for managing all activity of the cache, including instruction fetches from the cache, loading cache lines from memory, and invalidating cache lines.

---

The controller is basically a finite state machine (FSM) with three states, where each state has sub-states. The FSM diagram is shown in Figure 23.



**Figure 23: Cache Controller Finite State Machine**

After reset, the cache controller is disabled. In this state, when the processor makes an instruction request, a 256-bit data item including the desired instruction is fetched from the memory, the requested instruction is selected from the incoming data, and placed onto the instruction bus. All the valid bits are also reset when the controller enters the disabled state.

The controller enters the normal state by software enabling of caching with a write to the cache enable bit in the PSW. In this state, two operations are possible: read and invalidate. During a read operation the controller performs an instruction fetch by comparing the tag portion of the supplied address with the tag of the appropriate line of the tag RAM. If they match and the valid bit is set, then the desired word is selected, placed onto the instruction bus, and the HIT signal is asserted. Otherwise, the HIT signal is negated, and the controller enters the memory service state. If the INV signal is high, then the valid bit of the cache line specified by the instruction address is reset if the tag of the address matches the tag of the line.



---

The memory service state is very similar to the disabled state. The only difference is that when the data is fetched from the memory, it is also written to the appropriate core RAM line, the tag is written to the corresponding line of the tag RAM, and the valid bit of that line is asserted.

## **7.5. Cache Control Instructions**

The only cache control instruction supported by theTA2\_SW instruction set is the ICLI (instruction cache line invalidate) instruction. This instruction supplies an address using the register-plus-offset addressing mode. If the address is found in the cache, the corresponding cache line is invalidated.

## **7.6. Deviations**

Initial implementations of theTA2\_SW architecture may not contain device ID information in the cache tags. The implication is that the icache should be enabled only when instruction fetch streams can be guaranteed to map to the same device. A jump to an address that maps to a different device should be preceded with a disabling of the icache to invalidate its contents and prevent aliasing of addresses to the same cache line from distinct device IDs. While these actions are sufficient for the general case, they may not always be necessary. For example, when booting, if ROM code is copied directly to corresponding addresses of an eDRAM it is not necessary to disable and invalidate the icache contents when jumping from ROM to eDRAM.

---

## Chapter 8 - Address Translation

---

### 8.1. Introduction

Parcels, application code, and data contain virtual addresses. To interpret these addresses, aTA2\_SW RISC processor must support a translation mechanism. However, the overhead of maintaining conventional page tables at each node is prohibitive. To simplify translation, we classify memory according to usage:

- *global memory* is composed of contiguous segments distributed across nodes, visible to applications running on any node.
- *dumb memory* is a region of a node's memory allocated to some other entity and untouched by local node processing.
- *local memory* is a region of a node's memory used exclusively by node routines. This rule is excepted during initialization when the Master RISC Node, or another system boot process, loads node software.

A node must be able to rapidly determine if an address is located in its own memory, and if so, find the physical address. To condense translation information, we use *segments*, each of which is defined by segment registers containing a base address and size. The local memory region is partitioned into eight segments in theTA2\_SW architecture, although this number could change in futureTA2\_SW implementations. Like pages in a conventional system, the segment descriptors are generic, and have meaning only when assigned by system software. For example, a logical allocation of the eight segments would be to assign one segment for each of the following:

1. Kernel code
2. Kernel data
3. Kernel stack
4. Kernel parcel buffer
5. User code
6. User data
7. User stack
8. User parcel buffer

Remote addresses are translated via the concept of a home node, which is guaranteed to have the translation. In addition to the local segments, a node maintains translation information for its resident portion of the global memory, as well as for any remote data for which it is the home node. The major advantages of this approach are that translation may be accomplished rapidly, and translation information on each node scales well.

The primary functions of the node address translation unit are to translate virtual addresses to physical addresses for those accesses which are locally resident and to provide access protection. The types of accesses generated by a processor that require translation include instruction fetches and data accesses to memory or memory-mapped devices such as parcel buffers, generated by load or store instructions.

Given the simplicity of the address translation scheme discussed above, very little hardware support is needed to effect efficient translation. A segment base address register and limit register is needed for each of the eight local segments. Also, one virtual base, limit, and physical base register are needed for each resident global segment. The initialTA2\_SW architecture provides four sets of global segment registers, although alternative architectures could provide more. The address translation unit contains no direct support for home node translation,

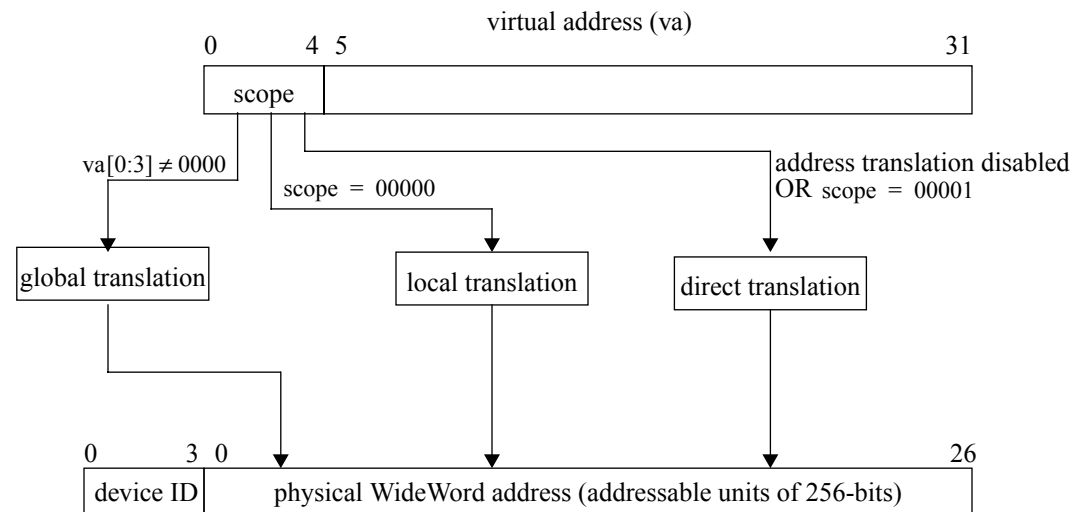
## 8.2. Address Translation Mechanisms

although the preferred system programming is such that the global segments resident on a node form the portion of global memory for which that node is the home node. If this is not the case, address faults invoke system software which performs the home node translation.

TheTA2\_SW RISC processor provides 4 Gbytes of virtual address space accessible to kernel and user applications via segments. Segment sizes can range from 256 bytes (minimum allowable segment size (MASS)) to the maximum amount of physical memory available to a node. The initial architecture supports a maximum segment size of 16 MBytes. Every segment size must be  $2^n$  MASS's, and the base address for each segment must be aligned to a value that is a multiple of the segment size. Given these stipulations base and limit register values are assumed to be in units of MASS's, resulting in 24-bit base address and 16-bit limits. Each virtual address generated by the processor is 32 bits wide, and the resulting physical address generated by the address translation unit contains a 4-bit device ID and 27-bit WideWord address, consistent with theTA2\_SW node interconnect specification. As indicated in theTA2\_SW node interconnect specification, lane enable signals are used for any data access that is less than 256 bits.

The processor address translation unit supports three main types of address translation:

- direct address translation
- local address translation
- global address translation



**Figure 24: Address Translation Types**

Figure 24 shows the three main address translation mechanisms provided. When the address translation unit is disabled, direct address translation occurs, and the address translation unit will not generate any exceptions. In this case, the device ID of the resulting physical address is formed from bits 5 through 8 of the virtual address, and bits 9 through 26 are zero-padded from the left to form the 27-bit WideWord address needed by theTA2\_SW node bus. (Therefore, when the ATU is disabled or the direct translation mode is invoked, the addressable space

---

is only 128 MB.) Also, as previously noted, if the address of the access is not 256-bit aligned, then lane enable signals consistent with the TA2\_SW node interconnect specification are generated

If address translation is enabled, then the scope field of the virtual address must be inspected to determine what type of translation should be used. In the initial architecture, the scope field is the most significant five bits of the virtual address VA. If this 5-bit value is zero, then local translation is used. If the scope field equals binary value 00001, i.e., the virtual address falls in the range of 0x08000000 to 0x0FFFFFFF, direct translation is used to generate the physical address; however, unlike the mode where address translation is disabled, an exception can be generated in this case if access privileges are violated. By definition the address region 0x08000000 to 0x0FFFFFFF is a supervisor-level region. Therefore, any user-level attempt to access this region while address translation is enabled will trigger an exception. Lastly, if any of the four most significant bits of the virtual address are non-zero, i.e.,  $a[0:3] \neq 0$ , then global translation is used.

Figure 25 shows the steps involved in local address translation. The 3-bit index field of the virtual address is used to select a set of local segment registers for the translation. The device ID entry of the selected segment is simply passed on to the device ID field of the physical address. The segment base is simply bitwise-ORed with the zero-padded offset of the virtual address to form the 27-bit WideWord physical address. The specified segment limit register is also accessed and manipulated in conjunction with the offset to determine if the virtual address is valid. More information on protection is given in the next section.

### Figure 25: Local Address Translation

Figure 1 shows the steps involved in global address translation, which is a reverse address translation style. In this case, the address is checked to see if it is mapped locally by simply ensuring that the address is within the range specified by a valid set of the global segment base address and limit registers. The hardware does not protect against overlapping global segments, i.e., system software must set up the global segment registers appropriately so that any global virtual address is contained in at most one global segment. The multiple sets of glo-

---

bal segment registers are checked concurrently to see if any one of them should be used for the translation, similar to a fully associative cache. If there is no match, a translation exception occurs. More detail on this matching and protection checking is given in the next section. If there is a match, the virtual address is simply translated into a physical address by passing the device ID entry of the matching segment to the appropriate field of the physical address. Also, the physical WideWord address is formed through a bitwise-OR of an offset with the global segment physical base register of the matching global segment. The offset is formed by using the limit register of the matching segment to mask off the appropriate part of the virtual address.

**FIGURE 1.**

**Figure 1: Global Address Translation**

### **8.3. Memory Access Protection**

In addition to the translation of virtual addresses to physical addresses, the address translation unit provides access protection and bounds checking to ensure that the offset portion of an address is not outside the range of the segment. The 2 PR bits of a segment limit register specify the access protection mode for that segment. Table 5 shows the possible access modes and their corresponding encodings.

**TABLE 5. Segment Access Modes and Corresponding PR Bit Encodings**

Encoding of PR Bits	Supervisor Privilege	User Privilege
00	RW (read-write)	RW
01	RW	RO (read only)
10	RW	none
11	RO	none

Each local segment limit register consists of a limit value, a valid bit, and the two PR bits. The first level of protection for local addresses is provided by ensuring that a valid set of segment registers is used. If the V bit of the selected local segment is not asserted, an unmapped access exception occurs (refer to Chapter 9). The second level of protection is provided by the PR bits. If the processor mode (supervisor or user) and access type (read or write) are not allowed by the PR bit setting of the selected segment, an invalid access exception occurs (refer to Chapter 9). The final level of protection for local addresses is provided with bounds checking. The limit value of the specific segment is used to inspect bits in the virtual address offset to ensure that the offset has not exceeded the segment size. If the segment size is exceeded, an unmapped access exception occurs (refer to Chapter 9). Assuming the limit value has been set according to the Implications section at the end of this chapter, an equation specifying the exception condition E is:

$$E = (va_8 \wedge \overline{\text{limit}[\text{index}]_0}) \vee (va_9 \wedge \overline{\text{limit}[\text{index}]_1}) \vee \dots \vee (va_{23} \wedge \overline{\text{limit}[\text{index}]_{15}})$$

Although the conditions for address translation exceptions for global virtual addresses are similar to that of local addresses, the mechanism is quite different due to the fully associative nature of the global segment hardware. Basically, if one of the four sets of global segment registers does not “match” an attempted global address access, an exception occurs. A successful match occurs when a set of segment registers is valid, the PR bit setting allows the access type being attempted, and the address range specified by the global virtual base and limit encompasses the global address of the operation. An equation specifying the range match condition RM, where va is the virtual address and base is the contents of the global virtual base register, is:

$$\overline{RM} = (va_0 \oplus base_0) \vee (va_1 \oplus base_1) \vee \dots \vee (va_7 \oplus base_7) \vee (\overline{\text{limit}_0} \wedge (va_8 \oplus base_8)) \vee (\overline{\text{limit}_1} \wedge (va_9 \oplus base_9)) \vee \dots \vee (\overline{\text{limit}_{15}} \wedge (va_{23} \oplus base_{23}))$$

An unmapped access exception is triggered if there is no valid set of registers that pass the range match test. If there is a valid set of registers that passes the range match test, but the PR bits for that segment do not allow the attempted access, an invalid access exception occurs (refer to Chapter 9).

## 8.4. Address Translation Unit Instructions

The primary instruction specified by the instruction set which affects address translation operation is the MTATR (move to address translation register) instruction. The destination field of this instruction can be set to specify any local base register, local protection register, global physical base register, global limit register, or global physical base register. Since the contents of a GPR is the data source for an MTATR instruction, each of these address translation unit registers is defined to be 32 bits wide, although implementations may truncate some segment registers to optimize for the actual amount of physical memory present. Furthermore, each limit register is a concatenation of a limit value, a valid bit, and the two PR bits. The MTPR instruction is also used to enable/disable address translation by writing to the appropriate bit of the PSW register.

## 8.5. Implications

There are a number of stipulations implied for the address translation mechanisms described in this chapter to operate correctly. First, every segment size must be a power of 2 MASS's, and the base address for each segment must be aligned to a value that is a multiple of the segment size. Also, the limit value must be set to  $(2^n - 1)$  for a segment size of  $2^n$ , so that logic functions used for translation and protection checking work properly. Finally, the virtual-to-physical translation for code segments must not affect the 12 least significant bits so that instruction cache look-ups can proceed concurrently with translation. While stipulating that code segment base addresses must be some multiple of 4Kbytes is sufficient, it is not necessary, and less strict policies can be used to ensure the requirement is met.

---

The exception portion of the architecture assumes that instruction and data address translations are independent. Thus, the PSW contains two address translation enable bits (one for instruction addresses and one for data addresses). Likewise, the exception source word contains separate status bits for instruction and data translation exceptions (refer to Chapter 9). There are also implications for better performance. For example, to allow address translation for both instruction fetches and data fetches to proceed concurrently, the address translation hardware must be dual-ported.

---

## Chapter 9 - Exceptions

This chapter defines the exceptions and exception-handling mechanism for the TA2\_SW RISC processor. Exceptions, arising from execution of node instructions, and interrupts, from other sources such as an internal timer or external interrupt signal, are handled by a common mechanism. For the most part this document will refer to both exceptions and interrupts as exceptions.

Traditionally RISC processors have had relatively primitive mechanisms for exception handling compared to CISC processors which may have multiple stack registers, extensive hardware-supported vectoring and priority-level controls for enabling exceptions. Even with these supporting hardware features, it is common to find problems of priority inversion and stack management errors in interrupt-service software. Errors in priority assignment are not easily fixed once cast in hardware. Exception handling hardware is difficult to implement and integrate with high-performance hardware.

The exception handling scheme for TA2\_SW has a modest hardware requirement, exporting much of the complexity to software, which is easier to mend. It does provide an integrated mechanism for handling hardware and software exception sources. Additionally, it provides a flexible priority assignment scheme which minimizes the amount of time that exception recognition is disabled. While the hardware design supports traditional stack-based exception handlers, we also outline a non-recursive dispatching scheme which uses TA2\_SW hardware features to allow preemption of lower-priority exception handlers using a mechanism which should be easier to debug.

### 9.1. Hardware-Vectored Exceptions

The TA2\_SW node processor must respond to a variety of exceptions due to internal instruction processing conditions and interrupts due to external stimuli. The processor has only three hardware-vectored exceptions. All other exceptions are dispatched by software with some hardware assistance. The exceptions are listed in descending priority order.

**TABLE 6. Hardware-Vectored Exceptions**

Exception	Vector Address	Notes
RESET	0x08000000 or 0x0A000000	If the “rom_present” input signal is asserted, the reset address is the base of the node FlashROM, 0x0A000000; otherwise, the reset address is the base of the node EDRAM, 0x08000000 (using addresses from the untranslated region, refer to Chapter 8)
Undefined Instruction (incl. BRK)	0x08000100	
Software-vectored exceptions	0x08000200	

Note that three of the vector addresses point to exception handler routines located at the start of node DRAM, so the node DRAM must be initialized and functional for any operation beyond system RESET.

All exceptions, other than reset and undefined-instruction exceptions, are vectored by hardware to the catch-all “software-vectored exception” handler, which examines the exception source word to perform a software-vectored dispatch to the appropriate exception handler.



## 9.2. Hardware Support for Hardware-Vectored Exceptions

The node processor has several privileged registers and a privileged instruction, RFE, used to return from exception handlers to normal processing.

All exception handlers operate in supervisor mode. The program counter and processor status words are copied to privileged temporary registers before exception processing is begun. The exception handling code runs in the same address map as the preceding code. Other state changes are performed at the exception handler if necessary. Other registers are set by specific exception conditions, e.g., MADR is set in the

**TABLE 7. Hardware State at Start of Exception Processing**

Register	Field	Value	Notes
PSW	MD	0	Mode is set to supervisor
PSW	EE	0	Exceptions disabled
PC		handler	Address of exception handler
FADR		old PC	Address of faulting instruction or next instruction
SSW		old PSW	Saved copy of prior PSW

event of a memory-access exception. The exception source word is set to indicate the cause of all but the reset and undefined-instruction exceptions, which are implicitly identified by the hardware vectoring to associated exception handlers. The exception source word and its associated enable mask register are discussed at more length in the “Software-Vectored Exceptions” section. Reset exceptions can not be disabled. All other exceptions may be disabled in aggregate by clearing bit 3 in the PSW. In addition all exceptions other than the Undefined Exception may be disabled selectively, by clearing a bit in the Exception Mask Register (refer to Section 9.5).

Upon completion of exception handling, the RFE instruction will copy the FADR to the PC and the SSW to the PSW to resume normal processing. Depending on the cause of the exception, the FADR may point to the instruction that caused the exception, if the exception prevented the instruction from completing, or to the next instruction in the code sequence, if the prior instruction did complete. For example, a memory access fault would load the FADR with the address of the load or store instruction which caused the access exception, while a timer interrupt or external interrupt would load the FADR with the next instruction to be executed. The exception handling code is responsible for adjusting the FADR as needed prior to executing the RFE instruction. Depending on the nature of the exception, the faulting instruction may be retried, for example a WideWord instruction after a lazy register save, or a memory access instruction after an address-translation adjustment.

The node processor provides four scalar system scratch registers to be used by exception handlers. Exception handling code requiring more registers are responsible for saving and restoring node processor registers as needed.

## 9.3. Hardware-Vectored Exception Descriptions

### 9.3.0.1. RESET (0x08000000 or 0x0A000000)

The external (system) RESET input causes instruction execution to begin at one of two possible reset addresses, depending upon the state of the *rom\_present* input signal to the processor. If *rom\_present* is asserted, indicating the processor has a FlashROM attached, the program counter will be loaded with 0x0A000000, the base address of the FlashROM in the untranslated address region, and instruction fetch will begin from this address when the processor is released from reset. If *rom\_present* is negated, the program counter will be loaded with 0x08000000, the base address of the eDRAM in the untranslated address region. Table 8 shows the state of the PSW register, processor status/control word, at reset.

### 9.3.0.2. Undefined Instruction (0x08000100)

This vector services all undefined instruction exceptions and also serves as the primary exception handler for breakpoint instructions. Breakpoint instructions are implemented by a software convention defining one or more undefined instruction opcodes as BRK<sub>x</sub>. For this type of exception to be recognized by the processor, bit 3 of the PSW must be set. Upon exception, the FADR register points to the address of the undefined instruction. To allow the BRK mechanism to debug exception handling code, we adopt the convention that SR3 is reserved exclusively for use by this exception handler, which does not use other scratch registers. This is not adequate to allow use of BRK prior to copying of FADR and SSW however. Also, for this exception to be recognized in exception handling code, exceptions must be re-enabled by setting

**TABLE 8. PSW State at RESET**

Bit	Field	Value	Notes
0	MD	0	Mode is set to supervisor
1	Unused	X	Reserved
2	IC	0	Instruction cache is disabled
3	EE	0	Exception recognition is disabled
4	WW	0	WideWord instruction processing is disabled
5	FP	0	Floating-Point Instruction processing is disabled
6 - 7	Unused	X	Reserved
8	IA	0	Instruction address translation is disabled
9	DA	0	Data address translation is disabled
10 - 31	Unused	X	Reserved

bit 3 of the PSW. To allow only undefined instruction exceptions, exceptions should be enabled while all bits of the Exception Mask Register (EMR) are cleared to disable all other exception types (refer to Section 9.5).

### 9.3.0.3. Software-vectored exceptions (0x08000200)

This vector provides the initial exception handling for all other exceptions and interrupts in the system. Recognition of this aggregate exception may be disabled by privileged code altering the PSW and is automatically disabled upon exception recognition, to remove any hardware requirement to support nested exceptions.

## 9.4. Software-Vectored Exceptions

Most exception sources are serviced by a software-vectored exception handler. Determination of the exception cause requires examination of the 32-bit exception source word, which constantly monitors hardware which may cause exceptions and also provides the ability for software to trigger exceptions.

Nested exceptions can be supported if the exception handler saves essential state, notably FADR and SSW, prior to reenabling exceptions. The software-vectored exception handling procedure supports nesting of exceptions for some potentially lengthy handlers by splitting the exception handler into primary and secondary parts. Primary exception handlers are non-interruptible except for reset. Secondary exception

---

	handlers may be interrupted by other exceptions. They may or may not be re-entrantly interrupted by other instances of the same exception type, depending on the handler code treatment of the mask register.
<b>9.4.1. Lightweight Exceptions</b>	<p>Lightweight exceptions are those which can be serviced completely within the primary exception handler, and do not require saving of transient exception state. Hardware disables further exceptions until reenabled by execution of RFE.</p> <p>An example of a lightweight exception is the timer tick exception, which increments a counter in memory. If the tick does not end a scheduling quantum, no further processing is required. If the tick does end a scheduling quantum, it triggers a quantum-expiration exception, but does no further processing itself.</p>
<b>9.4.2. Heavyweight Exceptions</b>	Heavyweight exceptions are those which cannot be serviced entirely within a primary exception handler. The primary exception handler saves necessary exception state in one of three locations. Temporary use is made of the system scratch registers. Processor context is saved, as necessary, in a register save area in a fixed-location memory area common to all primary exception handlers. Information specific to the particular exception, which is required for later processing by the secondary exception handler is saved in a fixed-location memory area specific to that particular exception type.
<b>9.4.3. Primary Exception Handlers</b>	<p>Primary exception handlers perform all of the processing for lightweight exceptions and the initial time-critical portion of heavyweight exceptions.</p> <p>The environment of primary exception handlers is highly constrained. They may use the system scratch registers SR0-SR3 freely but must save and restore any other GPRs. Primary handlers may call other routines conforming to the constraints, but must use the exception stack, which is located at the top of the kernel stack segment. Calling a subroutine in the primary exception handler environment requires initializing the stack pointer to the fixed top of the exception stack area. Primary handlers are written in assembly language.</p>
<b>9.4.4. Secondary Exception Handlers</b>	Secondary exception handlers perform the non-initial processing of heavyweight exceptions. They may not use the system scratch registers SR0-SR3, since exceptions are enabled during most of the execution of the secondary handler. Secondary handlers may be written in a restricted subset of the C language. Secondary handlers are written in a stylized form providing functions to suspend and resume their processing if preempted by higher priority exceptions.
<b>9.5. Hardware Support for Software-Vectored Exceptions</b>	<p>All software-vectored exception sources have an associated bit defined in the 32-bit exception source word, ESW, and corresponding bits in the exception-enable mask register, EMR, the exception set register, ESR, and the exception reset register, ERR. When a software-vectored exception is recognized, the global exception enable bit in the processor status word, PSW, is cleared, so that hardware events which cause changes to the ESW cannot trigger a nested exception. Reset exceptions may preempt primary exception handling code, but other exceptions will not be recognized.</p> <p>The exception source word is a 32-bit register recording exceptions initiated both by hardware and software sources. Hardware-source bits in the exception source word may be set to one by hardware conditions, such as a pbuf interrupt, while software-source fields are set by software writing a one to the corresponding bit location in the exception set register. Once set, a bit in the exception source word can be cleared</p>

---

only by writing a one to the corresponding bit of the exception reset register. Although labeled registers, both ESR and ERR are really regis-

**TABLE 9. Exception-Related Registers**

Name	PR#	Description
Exception Source Word (ESW)	8	Specifies sources of xceptions
Exception Enable Mask Register (EMR)	9	Bitwise exception enabling mask, 1 = enabled
Exception Set Register (ESR)	10	Write 1 to set corresponding bit in source word, SW-source fields onl
Exception Reset Register (ERR)	11	Write 1 to clear corresponding bit in word register

ter-address triggering functions; ESR and ERR do not maintain any state. That is, a one written to any bit in either of these registers causes an immediate and one-time effect on the corresponding bit in the exception source word.

Bits in ESW are affected by hardware conditions and ESR and ERR actions regardless of settings of the exception enable mask register, EMR. The bits of EMR merely enable, or disable, corresponding bits of ESW to cause exceptions. Therefore, there is a global exception enable control via the exception enable bit in PSW and individually maskable controls for each bit of the ESW via the EMR.

The Exception Source Word has 32 possible hardware- and software-initiated exception sources. The priority of the sources decreases with increasing bit number.

**TABLE 10. Exception Source Word**

Exception Name	Initiator	Bit#	Description
Watchdog Timer	HW	0	
Unmapped Instruction Access	HW	1	Instruction access not within segment boundaries
Invalid Instruction Access	HW	2	Instruction access not permitted
Unmapped Data Access	HW	3	Data access not within segment boundaries
Invalid Data Access	HW	4	Data access not permitted
PBuf Interrupt	HW	5	
Reserved	SW	6	
Interval Timer	HW	7	TIMER (protected register 13) count expired
WideWord Not Available	HW	8	WideWord instruction attempted without enable
Floating Point Not Available	HW	9	Floating-point instruction attempted without enable
Address Fault Fix-up	SW	10	
Received Packet Processing	SW	11	
Send Error Processing	SW	12	
Reserved	SW	13	

**TABLE 10. Exception Source Word**

Exception Name	Initiator	Bit#	Description
FP Divide by Zero	HW	14	
FP Invalid	HW	15	Triggered by scalar FPU or FPSR
FP Unsupported Value	HW	16	Triggered by scalar FPU or FPSR (underfl w or overfl w)
FP Inexact	HW	17	Triggered by scalar FPU or FPSR
Context Swapper	SW	18	
System Call	HW	19	
Privileged Instruction Violation	HW	20	
Scalar Integer ALU Exception	HW	21	
WideWord Integer ALU Exception	HW	22	
PBuf doorbell processing	SW	23	
Integer ALU Fix-up	SW	24	
WideWord ALU Fix-up	SW	25	
Floating Point Fix-up	SW	26	
Reserved	SW	27	
Lock Buzzer	SW	28	May not be implemented
Thread Rescheduler	SW	29	
Thread Dispatcher	SW	30	
Return to User Mode	SW	31	Full register restore as necessary

## 9.6. Dispatch of Software-Vectored Exception Handlers

### 9.6.1. Dispatch to the primary handler

TheTA2\_SW exception handling mechanism requires little specialized hardware support and supports preemption of lengthy low priority handlers without requiring LIFO processing due to stack mechanisms. Dispatch is always to the highest priority exception handler. There is no possibility of pathological stack growth under high rates of exceptions. System overload due to design problems will manifest as overruns, which can be evident and recoverable, rather than stack explosion, which is typically obscure and fatal.

A new exception condition will be recognized if exceptions are enabled in the PSW and if the particular source is enabled by the mask register. The hardware begins execution of code at the software-vectored exception vector address. Exceptions are disabled in the new PSW. Since the primary handlers are non-recursive and run to completion, processor state can be saved to a reserved temporary area at a fixed address (rather than a true stack) as needed by the particular handler.

---

The exception source word is copied into a scalar GPR and the ELO instruction is used to encode the bit number of the leftmost (smallest numbered) set bit. This operation selects the highest priority source. The encoded source bit number is used as the index into a vector of handler addresses, and the processor branches to that primary handler.

### ***9.6.2. Completion of a primary handler***

The selected primary handler determines whether the exception is lightweight enough to be handled in the primary handler or whether additional processing must be deferred to the secondary handler.

If the primary handler can complete the exception processing, it does so and then restores the saved GPRs and status before reenabling exception recognition by executing the RFE instruction. Prior to completion it will reset its associated exception source bit.

If the primary handler cannot complete the exception processing, it will copy the necessary state to a structure associated with its secondary handler, and set the bit associated with the secondary handler by writing to the exception set register. After restoring saved GPRs and status and resetting its source bit, it reenables exception recognition by executing the RFE instruction. The highest-priority exception source will subsequently be recognized and begin exception processing. This may be the secondary handler just scheduled or a higher priority hardware or software exception handler.

### ***9.6.3. Dispatch of a secondary handler***

The initial phase of the software vectoring of a secondary handler is the same as a primary handler. After the handler branches to the specific secondary handler code, the secondary handler is required to perform more elaborate state saving due to the possibility of preemption by higher-priority sources. The first phase of the secondary handler runs with exceptions disabled.

When a secondary handler begins execution it installs a pointer to its environment structure in the privileged register SR2. If the prior value of SR2 is zero, it is not preempting another secondary handler. If the prior value is nonzero, it is preempting another lower-priority secondary handler. To preempt, the current handler saves the state of the prior secondary handler by calling its suspend routine, the address of which is at a fixed offset within the environment. The suspend routine copies the necessary state into the environment and returns. The environment will typically hold only one instance of a given type of suspended secondary handler. This means that while exceptions can interrupt and preempt secondary handlers of a different type, we don't support reentrant handling of multiple exceptions of the same type. While it is a straightforward extension to support a per-type stack or queue of multiple exception instances, in most circumstances the inability to complete exception processing prior to encountering a subsequent exception of the same type reflects an underlying system-design problem.

Secondary handlers are coded to record essential state at periodic intervals. In effect, a secondary handler stores a checkpoint record of its progress in its environment with sufficient detail to allow processing to resume in the event of a preemption. A technique sufficient to maintain atomicity is to "double buffer" a structure with essential information and "flip" between the consistent and working copies with a write to an index or pointer variable. Code progress can be recorded by using a state variable for a software state machine or by updating function pointers.

In contrast to a traditional stack-based system, which keeps activation records on a stack which must be unwound in a LIFO order, the TA2\_SW RISC dispatch scheme records the activation of the handler by a bit in the exception source vector, while storing the associated saved state of preempted handlers in handler-specific environment structures. This ensures completion of handlers in priority order without requiring hardware support of multiple priority levels for exception recognition. It may also reduce the amount of saved state. The handler itself can be coded to record the bare minimum of state to allow a resumption, rather than being forced to assume the worst case and save entire register sets which may or may not have been altered. This is particularly significant for the large register sets of the TA2\_SW node.

The "checkpointed" exceptions scheme is much easier to debug via an interactive debugger or memory dump, since the state of each active exception handler is recorded at fixed locations in a form that may be conveniently examined as a high-level structure. This is in contrast to

---

a preemptive stack-based record, where the states of several handlers may be distributed in their lowest-level bindings across large chunks of stack at highly variable locations.

***9.6.4. Completion of a secondary handler***

The secondary handler completes by reinitializing its checkpoint record to its starting state, resetting its associated exception source word bit, and executing an RFE. If no other exception is recognized, the lowest-priority software exception will restore all disturbed register states and return to user-mode code.

# Test Article 2 Software (TA2\_SW) Article Memory Interface Description

October 1, 2009

University of Southern California  
Information Sciences Institute



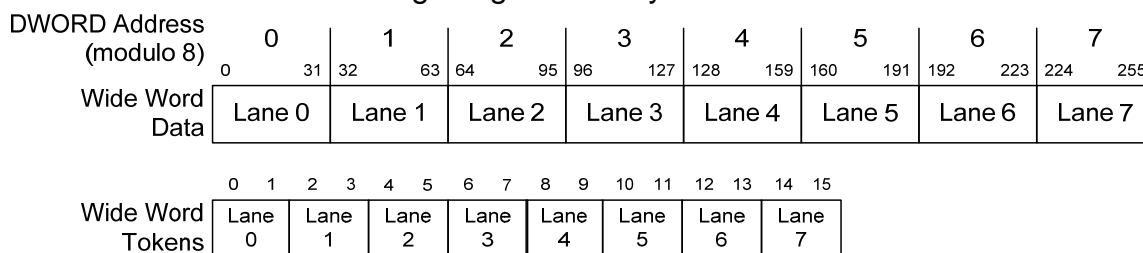
**SCOPE.** This document describes the TA2\_SW article memory interface (also referred to as the node bus interface as its original application served as a generic bus interface).

**Item Description.** The Node bus is a high performance 256-bit bus with de-multiplexed address and data. The Node bus is used as a memory access bus, an interface to input/output (I/O) devices, and/or a command and control bus. This bus supports a command/address & data protocol.

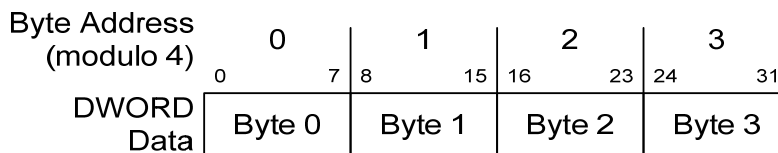
**Node Bus Overview.** The Node bus is designed as point to point ring interface. The ring consists of a single unidirectional channel with an optional second unidirectional counter-rotating channel. A bidirectional ring configuration potentially allows lower latency for reads and writes between communicating devices on a node ring.

Conventions:

The Node Bus uses big-endian notation in its numbering of bits and DWORDs. However, since the Node Bus address is not a byte or a DWORD address, the Node Bus is not big-endian per se. None the less, TA2\_SW is a big-endian design, and it is recommended that devices which internally use byte or DWORD addresses, map those addresses to the Node Bus using a big-endian style.



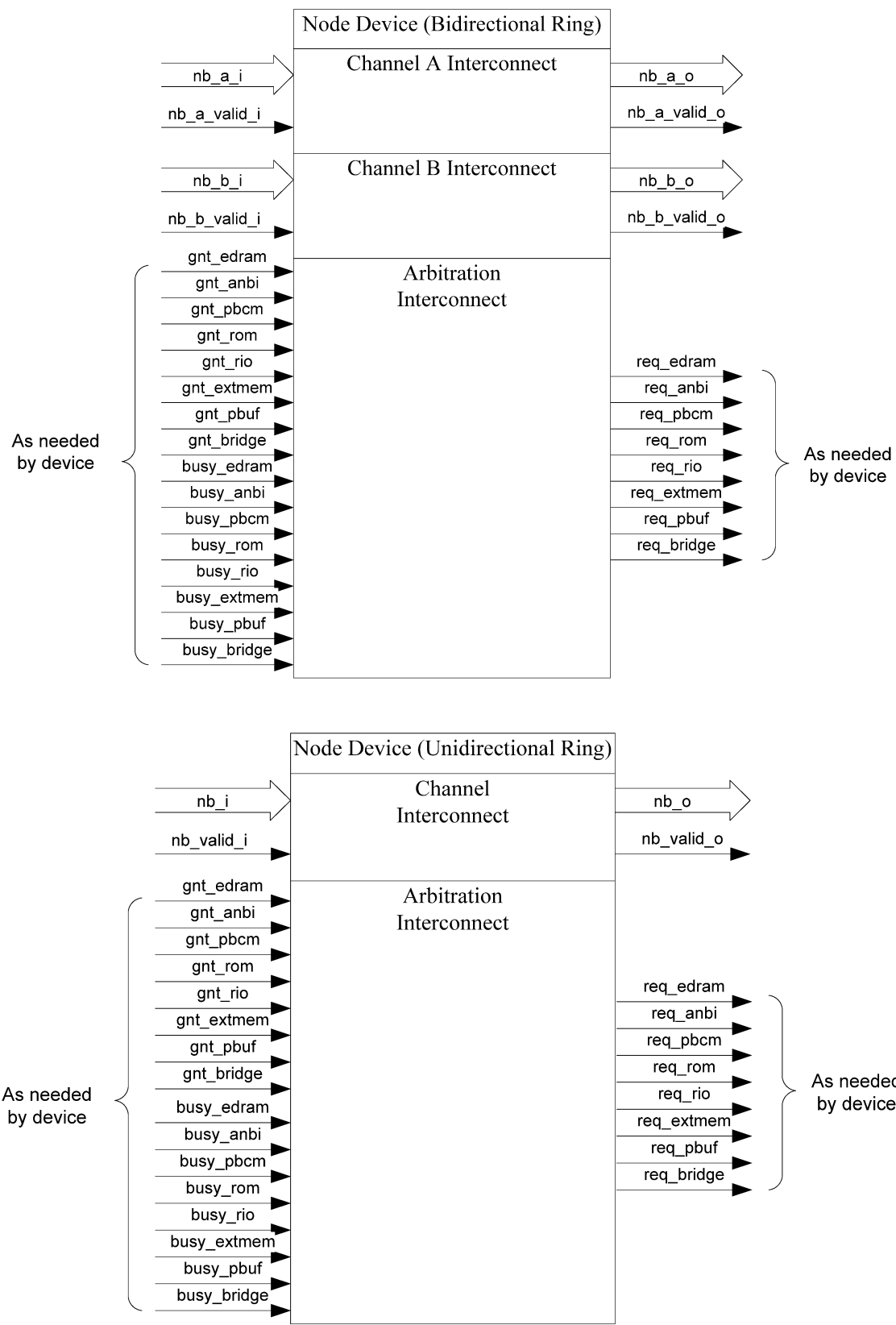
**Figure 1.3-1 – Suggested DWORD addressing within a Wide word**



**Figure 1.3-2 – Suggested Byte addressing within a DWORD**

## Signal Definition

Figure 2-1 shows the generic signals for unidirectional and bidirectional ring architectures in functional groups. Refer to a component's RTL description for more detail on which of these signals are necessary for a particular component.



**Figure 2-1 – Generic node bus interface signals**

## Signal Descriptions.

### Node Bus Signal Interfaces

A unidirectional ring Node Bus shall support the following signal interfaces:

nb_i	in	Node Bus Input – Channel input to device router. (Packet Structure as defined below)
nb_o	out	Node Bus Output – Channel output from device router. (Packet Structure as defined below)
nb_valid_i	in	Node Bus Valid Packet In – Channel Transaction Valid to Device Router.
nb_valid_o	out	Node Bus Valid Packet Out – Channel Transaction Valid from Device Router.

**Table 2-1-1 – Unidirectional Ring Node Bus Signals**

A bidirectional ring Node Bus shall support the following signal interfaces:

nb_a_i	in	Node Bus Input A – Channel A node input to device router. (Packet Structure as defined below)
nb_a_o	out	Node Bus Output A – Channel A node output from device router. (Packet Structure as defined below)
nb_a_valid_i	in	Node Bus Valid Packet In A – Channel A Packet Valid to Device Router.
nb_a_valid_o	out	Node Bus Valid Packet Out A – Channel A Packet Valid from Device Router.
nb_b_i	in	Node Bus Input B – Channel B node input to device router. (Packet Structure as defined below)
nb_b_o	out	Node Bus Output B – Channel B node output from device router. (Packet Structure as defined below)
nb_b_valid_i	in	Node Bus Valid Packet In B – Channel B Packet Valid to Device Router.
nb_b_valid_o	out	Node Bus Valid Packet Out B – Channel B Packet Valid from Device Router.

**Table 2-1-2 – Bidirectional Ring Node Bus Signals**

Node Bus Packet Structure: The Node Bus packet structure is shown below.

Signal Name	Applicable Transactions	Notes																											
cmd(0:1)	All	Node bus command input (Note: in this context only “command” refers to both command and reply transactions): 00 : Read Command 01 : Read Reply 10 : Reserved 11 : Write Command																											
te	Read and Write command	Token Enable: Used for devices that can optimize away reading or writing of tokens. 0 : Tokens Disabled 1 : Tokens Enabled For writes with te = 1, tokens follow the lane enables (i.e. tokens are written for those lanes that are enabled). If the tokens are enabled and all data lanes are disabled, then all the tokens are written without the data. Devices incapable of handling tokens can ignore ‘te’.																											
target(0:3)	All	Target ID input: See section 3.1.9																											
source(0:3)	All	Source ID input: See section 3.1.9																											
bksz(0:1)	Read Command and Read Reply	Read command Block Size input: 00 : Reserved 01 : Single Wide Word Read 10 : Double Wide Word Read 11 : Quad Wide Word Read Read Reply: Sequence number (00 for first reply wide word, 01 for second, 10 for third, and 11 for fourth)																											
addr(0:26)	All	Wide Word Address input: address of a 272-bit wide word.																											
data(0:255)	All	Wide Word Data input: 256-bits of data distinguishable as eight 32-bit DWORDS. For read commands bits 0-26 contain the reply address; bits 27-255 are invalid.																											
token(0:15)	Read Reply and Write command	Wide Word Token input: 16-bits of tokens distinguishable as eight 2-bit tokens																											
le(0:7)	Read Reply and Write command	Wide Word Lane enable inputs for writes. All 1’s for Read Reply transactions. <table border="1"> <thead> <tr> <th>Lane</th><th>Data</th><th>Token</th></tr> </thead> <tbody> <tr> <td>0</td><td>0:31</td><td>0:1</td></tr> <tr> <td>1</td><td>32:63</td><td>2:3</td></tr> <tr> <td>2</td><td>64:95</td><td>4:5</td></tr> <tr> <td>3</td><td>96:127</td><td>6:7</td></tr> <tr> <td>4</td><td>128:159</td><td>8:9</td></tr> <tr> <td>5</td><td>160:191</td><td>10:11</td></tr> <tr> <td>6</td><td>192:223</td><td>12:13</td></tr> <tr> <td>7</td><td>224:255</td><td>14:15</td></tr> </tbody> </table>	Lane	Data	Token	0	0:31	0:1	1	32:63	2:3	2	64:95	4:5	3	96:127	6:7	4	128:159	8:9	5	160:191	10:11	6	192:223	12:13	7	224:255	14:15
Lane	Data	Token																											
0	0:31	0:1																											
1	32:63	2:3																											
2	64:95	4:5																											
3	96:127	6:7																											
4	128:159	8:9																											
5	160:191	10:11																											
6	192:223	12:13																											
7	224:255	14:15																											

**Table 2-1-3 – Node Bus Packet Definition**

## Arbitration Pins.

Before transmission of a Read command or Write command packet, arbitration must take place to ensure the slave device has space to buffer the command packet. A master device on the Node Bus shall use the following request/grant/busy signals to the appropriate slave arbiter.

req_edram	out	<i>Node bus Request to EDRAM device.</i>
req_anbi	out	<i>Node bus Request to ANBI device.</i>
req_pbcm	out	<i>Node bus Request to Program Bus/CM device.</i>
req_rom	out	<i>Node bus Request to Flash/ROM device.</i>
req_rio	out	<i>Node bus Request to Rapid I/O device.</i>
req_extmem	out	<i>Node bus Request to External Memory device.</i>
req_pbuf	out	<i>Node bus Request to PBUF device.</i>
req_bridge	out	<i>Node bus Request to PCI Bridge device.</i>
gnt_edram	in	<i>Node bus Grant from EDRAM device.</i>
gnt_anbi	in	<i>Node bus Grant from ANBI device.</i>
gnt_pbcm	in	<i>Node bus Grant from Program Bus/CM device.</i>
gnt_rom	in	<i>Node bus Grant from Flash/ROM device.</i>
gnt_rio	in	<i>Node bus Grant from Rapid I/O device.</i>
gnt_extmem	in	<i>Node bus Grant from External Memory device.</i>
gnt_pbuf	in	<i>Node bus Grant from PBUF device.</i>
gnt_bridge	in	<i>Node bus Grant from PCI Bridge device.</i>
busy_edram	in	<i>Node bus Busy from EDRAM device.</i>
busy_anbi	in	<i>Node bus Busy from ANBI device.</i>
busy_pbcm	in	<i>Node bus Busy from Program Bus/CM device.</i>
busy_rom	in	<i>Node bus Busy from Flash/ROM device.</i>
busy_rio	in	<i>Node bus Busy from Rapid I/O device.</i>
busy_extmem	in	<i>Node bus Busy from External Memory device.</i>
busy_pbuf	in	<i>Node bus Busy from PBUF device.</i>
busy_bridge	in	<i>Node bus Busy from PCI Bridge device.</i>

**Table 2-1-3 –Node Bus Arbitration Signals**

**INTERFACE REQUIREMENTS.** This section defines the general operation of the Node Bus and provides a functional description of each interface signal.

Functional Description. The Node Bus supports transactions, which represent end-to-end operations from a source device to a target device. A packet is defined as the collection of address/data/control information of a transaction.

General Description. The Node Bus shall support command transactions consisting of write and read commands and reply transactions consisting of read replies.

Read operations are split into read command and one or more read reply transactions to allow data movement while waiting for returned read data. Each read command transaction specifies a base address from which 1, 2, or 4 wide words are read (sequentially starting from the base address). These words are then returned in order in 1, 2, or 4 read reply transactions. Any number of system clocks can occur between the read command and the corresponding read reply/replies. Read command transactions are always wide word reads. The block size field within the read command transactions specifies the number of words to be read. The block size field within the read reply transactions specifies the sequence number of the word being returned.

Write command transactions each contain one wide word. Write command transactions allow the master device to set any to all of the corresponding lane enables associated with each of the eight 32-bit data (and corresponding 2-bit token) allowing none, single, multiple, or all 32-bit data words to be written in a single cycle. All data may be written with or without tokens. Tokens may be written without the corresponding data being written if all lane enables are deasserted.

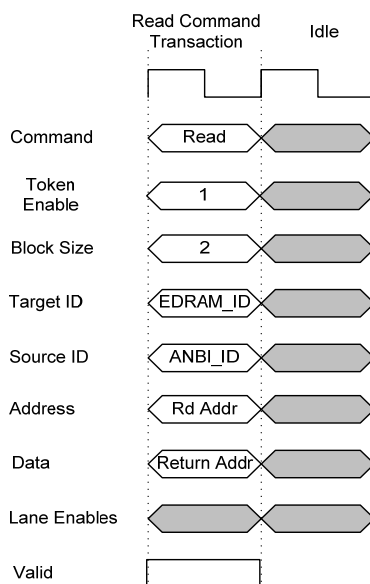
Command transactions are arbitrated while reply transactions are not. Command transactions targeted to a device that cannot guarantee space upon receipt are not granted in order to guarantee that all transactions inserted on the bus can be removed by the target. It is required that a device issuing a read command can sink replies without overflow.

Node Bus Operation. Packets are passed from device to device around a Node Bus channel. The valid signal is used to differentiate packets from idle cycles. The Node Bus works on the principle that all packet transfers complete in a single cycle and the bus is never stalled. Packets on the Node Bus have priority over packets waiting insertion onto the Node Bus. All inserted packets must wait until a cycle when no valid packet already on the Node Bus needs to be passed to the next device. The following paragraphs provide example transactions that occur on the Node Bus.

Read Command Transaction. In the example of figure 3.1.2.1-1, a typical read command transaction follows the format as follows:

- Command: Command of “00” indicates a Read Command.
- Token Enable: Tokens enabled
- Block Size: Read of two wide words requested.
- Target ID: EDRAM is being read
- Source ID: ANBI is requesting read (used for the read reply).
- Address: The address lines contain the address of the wide word to be read.
- Data: Contains the reply wide word address to be used by the slave for the reply transaction.
- Token: Don't care for a read command transaction.
- Lane Enables: Don't care for a read command transaction.
- Valid: The valid discrete is asserted to indicate a valid Node Bus packet.

DWORD accesses are not supported by read commands, and entire wide words shall be returned during read replies with all lane enables asserted. It is the requester's responsibility to extract the any desired DWORDS from the returned wide word.

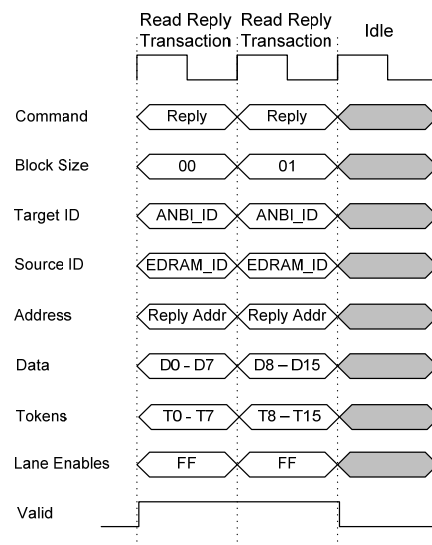


**Figure 3.1.2.1-1 – Example Node Bus Read Command Transaction**

Read Reply Transaction. A device inserts its reply transaction(s) corresponding to the

read command. In the example of figure 3.1.2.2-1, a typical read reply transaction follows the format as follows:

- Command: Command of “01” indicates a Read Reply.
- Block Size: The Block Size is “00” for the first reply word and “01” for the second.
- Target ID: Reply is sent back to ANBI (was Source ID during the read command transaction).
- Source ID: Reply is from EDRAM
- Address: The address lines contain the wide word read reply address contained in the data field of the corresponding read command.
- Data: Reply data.
- Tokens: Tokens corresponding to reply data
- Lane Enables: All lane enables are asserted since all read replies return a full wide word
- Valid: The valid discrete is asserted to indicate a valid Node bus packet.

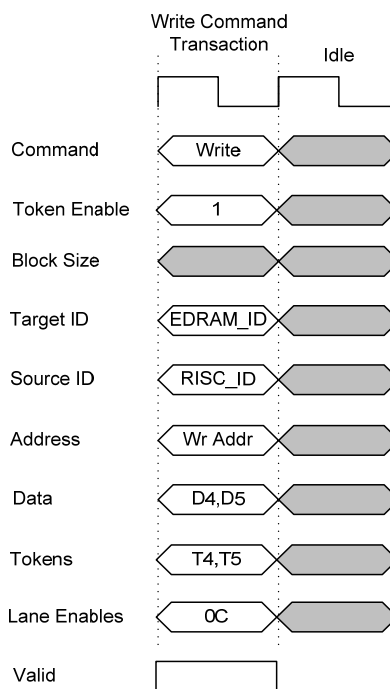


**Figure 3.1.2.2-1 – Example Node bus Read Reply Transaction**



Write Command Transaction. In the example of figure 3.1.2.3-1, a typical write transaction follows the format as follows:

- Command: Command of “11” indicates a Write.
- Token Enable: Indicates that tokens should be written along with data
- Block Size: Not valid.
- Target ID: Write to EDRAM
- Source ID: Write from RISC
- Address: The address lines contain the address where data should be written.
- Data: Write data
- Tokens: Write tokens
- Lane Enables: Indicates that DWORDS and Tokens 4 and 5 are to be written
- Valid: The valid discrete is asserted to indicate a valid Node bus packets.



**Figure 3.1.2.3-1 – Example Node Bus Write command Transaction**

## Pipelining

Each Node Bus device shall register packets at the input before being processed.

## Packet Insertion

A packet on a channel input of a device shall be passed to the corresponding channel output on the next cycle if the target ID does not match the device ID.

A packet may only be inserted on the channel output of a device if all of the following conditions are met:

- during the previous cycle an idle cycle or a packet destined for the device was received at the corresponding channel input
- the packet is a reply; or the packet is a command and a grant for the command transaction has been received from the corresponding slave arbiter (See 3.1.10)
- if the packet is a read command, the device can guarantee that it can receive all the read replies generated by the command without loss

An idle cycle shall be placed on the channel output of the device if no packet is output.

If a bidirectional ring is being used, packets shall be inserted on a channel based on a static routing vector (indexed by target ID.)

Note: if a bidirectional ring is used, zero, one, or two packets may be inserted on the same clock cycle (adhering to the insertion rules stated above.)

## Packet Receipt

A packet on the channel input of a device shall be passed to the device on the next cycle (removed from the channel) if the target ID matches the device ID.

Note: if a bidirectional ring is being used, zero, one or two packets may be received by the device on the same clock cycle (adhering to the removal rule stated above).

Deadlock Avoidance: To avoid deadlock, each device attached to a Node Bus must additionally obey the following requirements:

In order to execute or complete the execution of any received command, a device may not require that a command transaction is inserted into the

Node Bus from which it was received (either directly by the device itself, or indirectly via another device.)

In order to execute or complete the execution of any received reply, a device may not require that a command or reply is inserted into the Node Bus from which it was received (either directly by the device itself, or indirectly via another device.)

**Ordering.** The ordering of transactions on the Node Bus shall obey the following rules:

Between any source/target pair, commands must be processed by the slave device in the order in which they were generated by the master device.

Between any target/source pair, replies must be processed by the master device in the order in which they were generated by the slave device.

**Node Bus Cycles.** The Node Bus runs at the system clock rate (reference timing diagrams in Figures 3.1.2.1-1 through 3.1.2.3-1).

**Source/Target ID Codes.** The Source & Target ID shall identify the source and destination of the Node Bus transaction. Table 3.1.9 defines the allocation of Source/Target IDs. On replies, devices simply echo the Source ID bits received on the request packet.

Source/Target ID	Device
0000	EDRAM Device
0001	RISC Device
0010	ANBI Device
0011	Program Bus/CM Device
0100	Flash/ROM Device
0101	Rapid I/O Device
0110	External Memory Device
0111	Reserved
1000	PBUF Device
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	PCI Bridge Device

**Table 3.1.9 – Node bus Source/Target ID**

## Slave Arbitration

Each slave shall have an arbiter.

Each master shall send a request to the corresponding slave arbiter before it attempts to insert a command transaction on a channel.

A master may pre-request for one or more transactions from a slave arbiter (make a request and not send a corresponding command transaction for an indefinite period of time).

Each slave shall send a grant to a requesting master if and only if it can guarantee to receive another command transaction from the requesting master without loss. A grant which has been sent, and for which a corresponding command transaction has not yet been received is termed an outstanding grant.

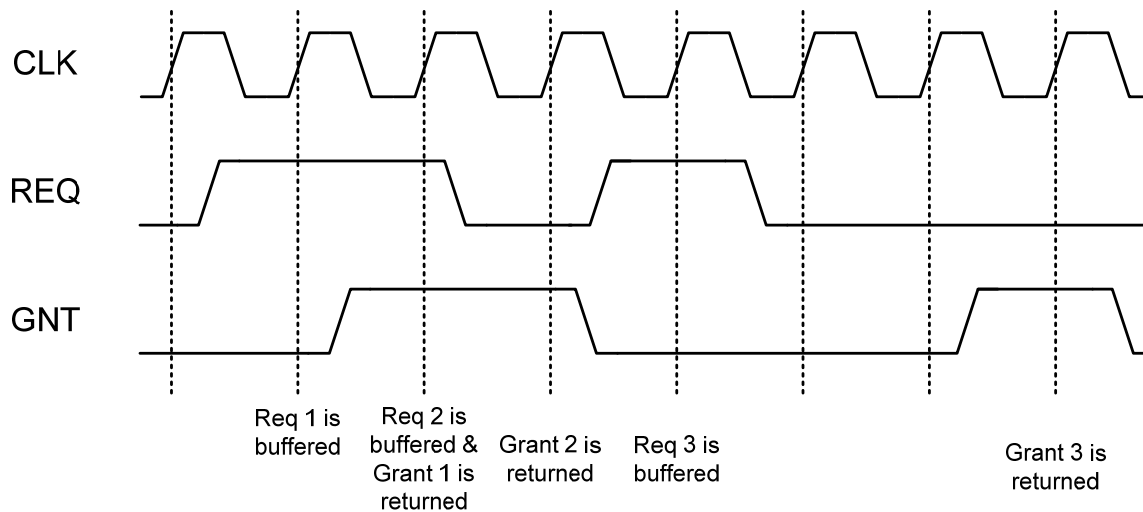
At all times, at least one of the following statements shall be true for each slave:

- the slave is processing a command transaction or have one or more command transactions waiting to be processed (a request transaction which has been removed from the bus, but for which the execution of the command is not yet complete)
- for some master, the slave has one more outstanding grant than the maximum number of pre-requests the master makes
- the slave is able to receive at least one more command transaction without loss (i.e., able to make a new grant)

## Arbitration Interface and Signal Description.

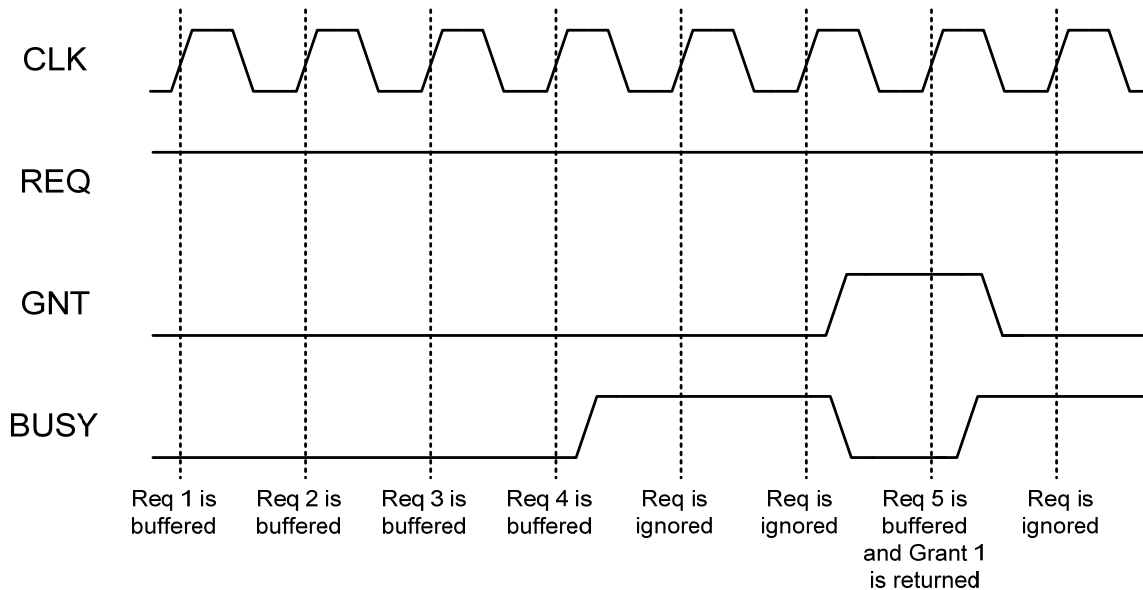
Each master device shall interface with a Node Bus slave device using a set of request/grant/busy signals (see Table 2-1-4) to the slave device's arbiter. The "req\_xxxx" signal represents the request from master device to send a command transaction to the slave device. The "gnt\_xxxx" signal indicates that the slave device can accept a command transaction from the master device. The "busy\_xxx" signal indicates the slave cannot accept new requests from the master device. Since the TA2\_SW RISC processor is never a slave and the EDRAM used for the testbench simulations is never a master, the testbench operation is straightforward.

Node bus slave arbiters shall accept pulsed requests and issue pulsed grants. (i.e., requests and grants held high for 2 or more clocks indicate multiple requests and grants). See figure 3.1.11-2.



**Figure 3.1.11-2 Pulsed Request / Grant Example**

Node bus slave arbiters shall buffer a non-zero number of requests on each request port. A busy signal shall be generated on that port whenever the arbiter will not buffer additional requests. This busy signal shall indicate to the requestor that requests made while busy is asserted will be ignored. Note that any request made during the first clock that busy is asserted has been ignored and must be reissued after the arbiter de-asserts busy.



**Figure 3.1.11-3 Request and Busy Signal Example**

## Appendix II. Glossary

Term	Definition
Channel	A set of connections forming a unidirectional node bus ring
Command	Arbitrated transaction type consisting of read and write commands
Device	A source and/or target on a node bus ring
DWORD	A 32-bit value
Master	A node bus device capable of sourcing <i>command</i> transactions.
Node Bus Idle Cycle	Occurs when the valid bit is deasserted on the Node Bus connection between two devices.
Packet	The collection of address/data/control information of a transaction
Reply	Unarbitrated transaction type consisting of read replies
Ring	A unidirectional or bidirectional, point-to-point cyclic connection of node bus devices
Slave	A node bus device capable of sinking command packets
Source	The device sourcing a transaction
Target	The device sinking a transaction
Token	A 2-bit value associated with a DWORD
Transaction	An end-to-end operation from a source device to a target device
Wide Word	256 bits of data + 16 bits of token

# Test Article 2 Software Article (TA2\_SW) RISC Processor Instruction Set Manual

**University of Southern California  
Information Sciences Institute**

---

---

(This page intentionally left blank.)



---

## Chapter 1 - TA2\_SW Instruction Set Overview

---

### Scalar Instruction Formats

As shown in Figure 1, the TA2\_SW scalar instruction uses a three-operand format to specify two 32-bit source registers and a 32-bit target register. For arithmetic/logical instructions using this format, there is also a **C** bit to indicate whether the current instruction updates condition codes. However, the **C** bit indicates signed/unsigned arithmetic for multiply/divide instructions, since these instructions never update condition codes by definition. In lieu of a second source register, a 16-bit immediate value may be specified, as shown in Figure 2.

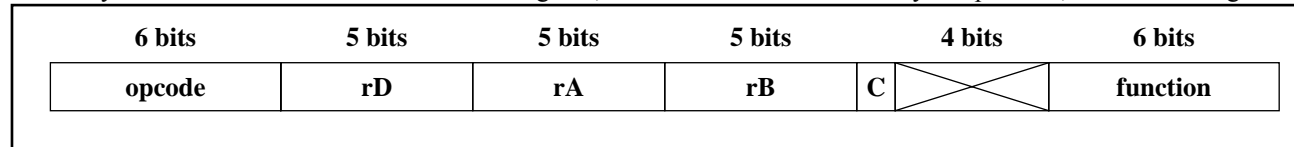


Figure 1 Format R for Scalar Register Operations

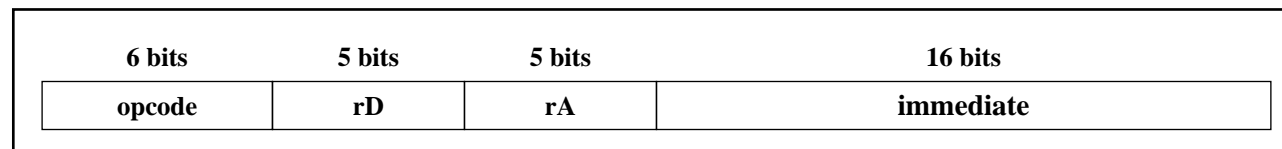


Figure 2 Format I for Scalar Immediate Operations

The branch instruction formats are shown in Figure 3. The branch target address may be PC-relative or calculated using a base register ORed with an offset. In both formats, the offset is in units of words, or 4 bytes, since instructions must be on a 4-byte boundary. Furthermore, the **L** bit specifies linkage, that is, whether a return instruction address should be saved in R31, referred to as a call instruction. Also, the **CCC** field specifies one of eight branch conditions: always, equal, not equal, less than, less than or equal, greater than, greater than or equal, or overflow. See the branch and call instruction descriptions for details.

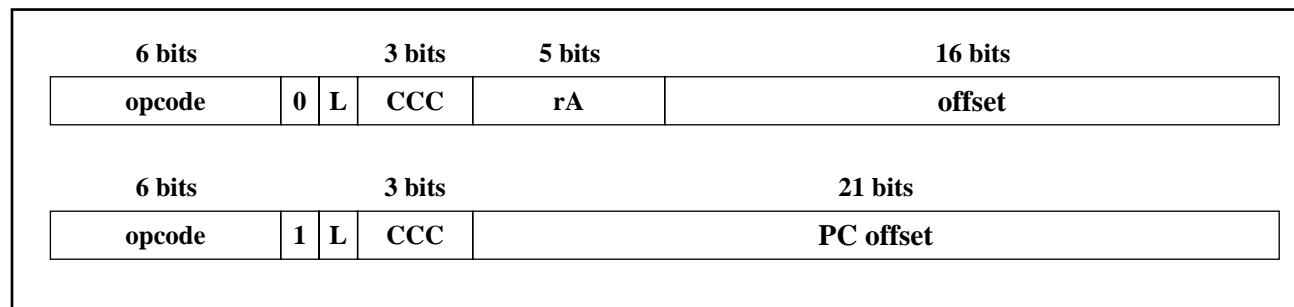


Figure 3 Format B for Branches

## WideWord Instruction Formats

Rather than using a dedicated 256-bit datapath as was designed in DIVA, TA2\_SW WideWord operations are executed in a morphable arithmetic cluster which may be configured for WideWord operations. However, the DIVA WideWord instruction set is largely preserved. As shown in Figure 4, “WideWord Arithmetic/Logical Format,” WideWord instructions follow the general form of scalar instructions. Additional control information is included to manage the data fields of the WideWord, and to modify the execution of the instruction. Figure 5 shows the format for transfers within the WideWord register file and across the scala, floating-point, and WideWord register files.

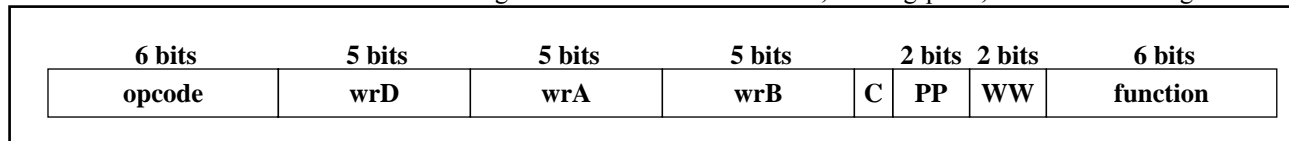


Figure 4 Format W for WideWord Arithmetic/Logical Operations

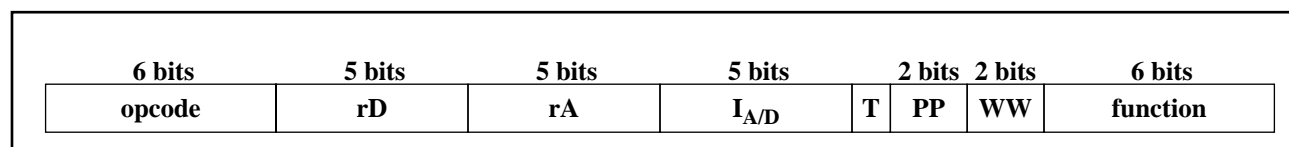


Figure 5 Format T for Wide-Word and Inter-Register File Transfers

The control fields are defined as follows:

### WW (width)

The **WW** field sets the width of the WideWord operands to eight, sixteen, or thirty-two bits, which primarily affects the shift operations and the configuration of the carry chain for additions and subtractions. For the merge instruction, these bits specify the condition on which the merge is based. The encoding of these bits is listed in the following table:

WW Value	Operand Width	Assembler Mnemonic
00	8 bits	b
01	16 bits	h
10	32 bits	w
11	Reserved	NA

### C (condition code enable)

The **C** bit indicates whether condition codes will be updated as a result of the current instruction’s execution in most cases. However, the **C** bit indicates signed/unsigned arithmetic for multiply, pack, and unpack instructions.

### PP (participation)

The **PP** field interacts with condition codes to control whether a computation is performed on a given data field. The participation field can specify that a data field participate always, only if a condition local to its own data field is true, only if the data field is the leftmost field with a condition that is true, or only if the data field is the rightmost field with a condition that

---

is true. The condition that is inspected for participation depends on the value of the **PM** (participation mode) register. Refer to the architecture document for more details. The encoding of the **PP** bits is listed in the following table:

PP Value	Participation Definition	Assembler Mnemonic
00	Always participate	a
01	Specified by local conditio	o
10	Reserved	NA
11	Reserved	NA

### ***T (type)***

The **T** bit governs whether the current instruction operates on a vector or scalar. Depending on the function, **rD** or **rA** may specify a WideWord register. In this case, the **T** bit specifies whether the current transfer instruction refers to the WideWord register as a whole vector or instead uses **I<sub>AD</sub>** to index a sub-field of the WideWord register.

### ***I<sub>AD</sub>***

Value to be used as an index when a sub-field of a WideWord is involved in a transfer. Depending on the function, this index field may be an immediate or a scalar GPR specifier. Also, **I<sub>AD</sub>** may be coupled with either **rD** or **rA** depending on the direction of the transfer as specified by the function.

### ***Tokens***

Each entry of the WideWord register file contains 256 bits of data and 16 bits of token information, 2 bits for each 32 bits of data. This is consistent with the association of tokens and data in the TA2\_SW streaming operations executed in the arithmetic clusters when configured for streaming mode (refer to the specification for the arithmetic cluster). The rationale for including tokens in the WideWord datapath is that the WideWord unit may be involved in processing streams stored in memory, and it is desirable for the tokens of the stream to be preserved for future streaming operations. To support this capability, nominally the tokens associated with the operand specifier by wrA are written to the token field of the operand specifier by wrD in any WideWord instruction. However, some TA2\_SW implementations may ensure token compliance for only WLD and WST instructions. For designs that implement the full token capability, tokens are not subject to participation. That is, the tokens of wrA will be written to wrD even if the participation effect masks off all data fields of wrD.

### ***Load/Store Buffers***

Some TA2\_SW/TA2\_SW implementations may include 256-bit buffers to improve performance for scalar integer and floating-point loads and stores. For such implementations, LD and FLD instructions first interrogate the load buffer(s) to see if the address contents of a load buffer matches bits 0 through 26 of the effective address of the instruction. If so, the 32-bit data to be fetched is retrieved from the load buffer, thereby avoiding a node interconnect and memory access. If not, the 256-bit WideWord containing the data is fetched from the node memory and loaded into the load buffer, and the appropriate 32-bit subfield is forwarded to the appropriate pipeline registers to continue the load operation. Similarly, ST and FST instructions attempt to complete via a store buffer. If the effective address matches that associated with a store buffer, the appropriate 32-bit subfield of the store buffer is written and the lane is marked dirty. If not, the current content of the store buffer is first flushed to memory, with the dirty bits serving as the lane enable signals (as specified by the node interconnect specification), and then the data and address of the ST or FST instruction are then written to the appropriate field of the store buffer. If the address of a ST or FST instruction matches that of a load buffer, the appropriate 32-bit subfield of the load buffer is also written. If the address of a LD or FLD instruction matches that of a store buffer, then any 32-bit subfield which are marked as dirty are forwarded from the store buffer -- the

other subfields are fetched from the node memory. The instruction set includes an explicit LDBI instruction to invalidate the load buffer, forcing a node memory access for the next LD or FLD instruction. Note: the LDBI instruction doesn't flush any store buffer -- therefore if the address of a subsequent LD or FLD instruction matches that of a store buffer, the data may be forwarded from the store buffer even after a LDBI. The instruction set also includes an explicit STBF to flush the store buffer to memory. The initial TA2\_SW implementation contains one 256-bit load buffer and one 256-bit store buffer.

### Condition Codes

The scalar condition code register, **CC**, consists of 5 bits. The first three bits of **CC** are set by an algebraic comparison of the result to zero; the other two bits have slightly more peculiar semantics. The condition codes have the **CC** bit labels and semantics as indicated below. Note that LT, GT, EQ, and CA condition codes are updated only if the current instruction has its condition code enable bit set. The OV condition

Condition Code	CC bit	Description
LT	0	This bit is set when the result represents a number strictly less than zero.
GT	1	This bit is set when the result represents a number strictly greater than zero.
EQ	2	This bit is set when the result represents a number equal to zero.
OV	3	This bit is set to indicate overflow has occurred during execution of an add or subtract instruction. This bit is not altered by any other instructions. In practice, the OV bit is set if the carry out of bit 0 is not equal to the carry out of bit 1 (assuming big Endian bit labeling).
CA	4	In general, the carry bit (CA) is set to indicate that a carry out of bit 0 occurred during execution of an add or subtract instruction. This bit is not altered by any other instructions.

code is updated for any scalar add or subtract operation, regardless of the condition code enable bit setting, and is sticky; that is, it is only cleared when the condition code register is read. They are accessed in conditional branch and call statements. Further, like any user-level special-purpose registers, they can be explicitly read and written with the MFSPR and MTSPR instructions, respectively. When accessed with these instructions, the 5-bit CC value is right-justified to the least significant bits of the 32-bit integer datapath.

The 32-bit LT, GT, EQ, OV, and CA registers of the WideWord datapath have analogous semantics to the corresponding condition code of the scalar datapath. For instance, each bit of the WideWord LT register is set if the result of its corresponding 8-bit datapath is negative. However, there are subtleties due to the configurability of the operand sizes. For example, if a WideWord instruction specifies that operands are to be treated as 32-bit values, the condition codes are grouped into eight groups of 4, where each bit of a group is updated with the same value to reflect a condition for the group's corresponding 32-bit result.

Similar to condition codes, the WideWord floating-point status register (FPSR - special-purpose register 15) may be updated to reflect exception conditions for WideWord floating-point operations. This register is a 32-bit register arranged in groups of 4 status conditions for each of the eight 32-bit floating-point units in the WideWord datapath. The 4 status conditions are: invalid (IV), inexact (IX), overflow (OV), and



TABLE 1. TA2\_SW Instruction Set

FUNC	DESCRIPTION	FUNC	DESCRIPTION	FUNC	DESCRIPTION
	<b>Scalar Instructions</b>		<b>WideWord Instructions</b>		<b>Branch Instructions</b>
<b>ADD</b>	Add	<b>WADD</b>	Add	<b>Bx</b>	Branch on scalar condition
<b>ADDE</b>	Add extended	<b>WADDE</b>	Add extended	<b>BAx</b>	Branch on all WideWord conditions
<b>ADDI</b>	Add immediate	<b>WSUB</b>	Subtract	<b>BNx</b>	Branch on no WideWord condition
<b>ADDIC</b>	Add immediate w/ condition codes	<b>WSUBE</b>	Subtract extended	<b>CALLx</b>	Call on scalar condition
<b>SUB</b>	Subtract	<b>WSUBU</b>	Subtract unsigned	<b>CALLAx</b>	Call on all WideWord conditions
<b>SUBE</b>	Subtract extended	<b>WMULES</b>	Multiply even signed	<b>CALLNx</b>	Call on no WideWord condition
<b>SUBU</b>	Subtract unsigned	<b>WMULEU</b>	Multiply even unsigned		<b>System Instructions</b>
<b>MUL</b>	Multiply	<b>WMULOS</b>	Multiply odd signed	<b>SYS</b>	System Call
<b>MULU</b>	Multiply unsigned	<b>WMULOU</b>	Multiply odd unsigned	<b>ICLI</b>	Instruction Cache Line Invalidate
<b>DIV</b>	Divide	<b>WAND</b>	And	<b>RFE</b>	Return from Exception
<b>DIVU</b>	Divide unsigned	<b>WNOT</b>	Bitwise inversion	<b>MTATR</b>	Move to address translation reg
<b>AND</b>	And	<b>WOR</b>	Or	<b>MFATR</b>	Move from address translation reg
<b>ANDI</b>	And immediate	<b>WXOR</b>	Xor	<b>MTPR</b>	Move to protected reg
<b>ANDIC</b>	And immediate w/ condition codes	<b>WSLL</b>	Shift left logical	<b>MFPR</b>	Move from protected reg
<b>NOT</b>	Bitwise inversion	<b>WSLLI</b>	Shift left logical immediate		<b>FPU Instructions</b>
<b>OR</b>	Or	<b>WSRA</b>	Shift right arithmetic	<b>FABS</b>	Floating-point absolute value
<b>ORI</b>	Or immediate	<b>WSRAI</b>	Shift right arithmetic immediate	<b>FADD</b>	Floating-point add
<b>ORIC</b>	Or immediate w/ condition codes	<b>WSRL</b>	Shift right logical	<b>FDIV</b>	Floating-point divide
<b>ORIS</b>	Or immediate shifted	<b>WSRLI</b>	Shift right logical immediate	<b>FLD</b>	Floating-point load
<b>XOR</b>	Xor	<b>WLD</b>	Load Reg from Mem	<b>FMUL</b>	Floating-point multiply
<b>XORI</b>	Xor immediate	<b>WST</b>	Store Reg to Mem	<b>FNEG</b>	Floating-point negate
<b>XORIC</b>	Xor immediate w/ condition codes	<b>WFABS</b>	Floating-point absolute value	<b>FST</b>	Floating-point store
<b>SLL</b>	Shift left logical	<b>WFADD</b>	Floating-point add	<b>FSUB</b>	Floating-point subtract
<b>SLLI</b>	Shift left logical immediate	<b>WFMUL</b>	Floating-point multiply	<b>FTI</b>	Floating-point to integer conversion
<b>SRA</b>	Shift right arithmetic	<b>WFNEG</b>	Floating-point negate	<b>ITF</b>	Integer to floating-point co version
<b>SRAI</b>	Shift right arithmetic immediate	<b>WFSUB</b>	Floating-point subtract		<b>Transfer Instructions</b>
<b>SRL</b>	Shift right logical	<b>WFTI</b>	Floating-point to integer conversion	<b>MVFF</b>	Move FPU to FPU
<b>SRLI</b>	Shift right logical immediate	<b>WITF</b>	Integer to floating-point co version	<b>MVFS</b>	Move FPU to scalar
<b>LD</b>	Load Reg from load buffer if possible	<b>WPRM</b>	Permute	<b>MVFW</b>	Move FPU to WW
<b>ST</b>	Store Reg to store buffer if possible	<b>WPRMI</b>	Permute immediate	<b>MVFWI</b>	Move FPU to WW, indirect
<b>LDBI</b>	Load buffer invalidate	<b>WMRG</b>	Merge based on condition codes	<b>MVSF</b>	Move scalar to FPU
<b>STBF</b>	Store buffer flush	<b>WPKS</b>	Pack using signed arithmetic	<b>MVSW</b>	Move scalar to WW
		<b>WPKU</b>	Pack using unsigned arithmetic	<b>MVSWI</b>	Move scalar to WW, indirect
	<b>Miscellaneous Instructions</b>	<b>WUPKL</b>	Unpack low-order byte/halfword	<b>MVWF</b>	Move WW to FPU
<b>MTSPR</b>	Move to special-purpose reg			<b>MVWFI</b>	Move WW to FPU, indirect
<b>MFSPR</b>	Move from special-purpose reg			<b>MVWS</b>	Move WW to scalar
<b>LOKL</b>	Lock Load			<b>MVWSI</b>	Move WW to scalar, indirect
<b>LOKS</b>	Lock Store			<b>MVWW</b>	Move WW to WW
<b>PROBE</b>	Probe address to determine locality			<b>MVWWI</b>	Move WW to WW, indirect
<b>ELO</b>	Encode leftmost one	<b>TKLD</b>	Token Load		
<b>CLO</b>	Clear leftmost one	<b>TKST</b>	Token Store		

## Alphabetical list of instructions

**TABLE 2. Encoding of TA2\_SW Instruction Set**

Instruction	Format	Encoding					
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
ADD	R	000011	rD	rA	rB	0XXXX	100000
ADDC	R	000011	rD	rA	rB	1XXXX	100000
ADDE	R	000011	rD	rA	rB	0XXXX	100001
ADDEC	R	000011	rD	rA	rB	1XXXX	100001
ADDI	I	100000	rD	rA	immediate		
ADDIC	I	100001	rD	rA	immediate		
AND	R	000011	rD	rA	rB	0XXXX	101000
ANDC	R	000011	rD	rA	rB	1XXXX	101000
ANDI	I	101000	rD	rA	immediate		
ANDIC	I	101001	rD	rA	immediate		
Bx	B	111111	00CCC	rA	offset		
Bx	B	111111	10CCC	PC-relative offset			
BAx	B	111100	00CCC	rA	offset		
BAx	B	111100	10CCC	PC-relative offset			
BNx	B	111101	00CCC	rA	offset		
BNx	B	111101	10CCC	PC-relative offset			
CALLx	B	111111	01CCC	rA	offset		
CALLx	B	111111	11CCC	PC-relative offset			
CALLAx	B	111100	01CCC	rA	offset		
CALLAx	B	111100	11CCC	PC-relative offset			
CALLNx	B	111101	01CCC	rA	offset		
CALLNx	B	111101	11CCC	PC-relative offset			
CLO	R	000011	rD	rA	00000	0XXXX	001001
DIV	R	000011	00000	rA	rB	0XXXX	100111
DIVU	R	000011	00000	rA	rB	1XXXX	100111
ELO	R	000011	rD	rA	00000	0XXXX	001000
FABS	R	000101	frD	frA	00000	0XXXX	000101
FABSC	R	000101	frD	frA	00000	1XXXX	000101
FADD	R	000101	frD	frA	frB	0XXXX	000000
FADDC	R	000101	frD	frA	frB	1XXXX	000000
FDIV	R	000101	frD	frA	frB	0XXXX	000111
FDIVC	R	000101	frD	frA	frB	1XXXX	000111
FLD	I	010000	frD	rA	offset		
FMUL	R	000101	frD	frA	frB	0XXXX	000110
FMULC	R	000101	frD	frA	frB	1XXXX	000110
FNEG	R	000101	frD	frA	00000	0XXXX	000100

**TABLE 2. Encoding of TA2\_SW Instruction Set**

Instruction	Format	Encoding					
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
<b>FNEGC</b>	R	000101	frD	frA	00000	1XXXX	000100
<b>FST</b>	I	010001	frD	rA	offset		
<b>FSUB</b>	R	000101	frD	frA	frB	0XXXX	000001
<b>FSUBC</b>	R	000101	frD	frA	frB	1XXXX	000001
<b>FTI</b>	R	000101	frD	frA	00000	0XXXX	000010
<b>FTIC</b>	R	000101	frD	frA	00000	1XXXX	000010
<b>ITF</b>	R	000101	frD	frA	00000	0XXXX	000011
<b>ITFC</b>	R	000101	frD	frA	00000	1XXXX	000011
<b>ICLI</b>	I	110011	00000	rA	offset		
<b>LD</b>	I	110000	rD	rA	offset		
<b>LDBI</b>	I	111000	rD	rA	offset		
<b>LOKL</b>	I	110110	rD	rA	offset		
<b>LOKS</b>	I	110111	rD	rA	offset		
<b>MFATR</b>	R	000000	rD	atrA	00000	XXXXX	000010
<b>MFPR</b>	R	000000	rD	prA	00000	XXXXX	000000
<b>MFSPR</b>	R	000001	rD	sprA	00000	XXXXX	000100
<b>MTATR</b>	R	000000	atrD	rA	00000	XXXXX	000011
<b>MTPR</b>	R	000000	prD	rA	00000	XXXXX	000001
<b>MTSPR</b>	R	000001	sprD	rA	00000	XXXXX	000101
<b>MUL</b>	R	000011	00000	rA	rB	0XXXX	100110
<b>MULU</b>	R	000011	00000	rA	rB	1XXXX	100110
<b>MVFF</b>	T	000100	frD	frA	00000	XXXXX	001010
<b>MVFS</b>	T	000100	rD	frA	00000	XXXXX	001001
<b>MVFW</b>	T	000100	wrD	frA	I <sub>D</sub>	TPP10	001000
<b>MVFWI</b>	T	000100	wrD	frA	R <sub>ID</sub>	00010	101000
<b>MVSF</b>	T	000100	frD	rA	00000	XXXXX	000110
<b>MVSW</b>	T	000100	wrD	rA	I <sub>D</sub>	TPPWW	000100
<b>MVSWI</b>	T	000100	wrD	rA	R <sub>ID</sub>	000WW	100100
<b>MVWF</b>	T	000100	frD	wrA	I <sub>A</sub>	00010	000010
<b>MVWFI</b>	T	000100	frD	wrA	R <sub>IA</sub>	00010	100010
<b>MVWS</b>	T	000100	rD	wrA	I <sub>A</sub>	000WW	000001
<b>MVWSI</b>	T	000100	rD	wrA	R <sub>IA</sub>	000WW	100001
<b>MVWW</b>	T	000100	wrD	wrA	I <sub>A</sub>	TPPWW	000000
<b>MVWWI</b>	T	000100	wrD	wrA	R <sub>IA</sub>	1PPWW	100000
<b>NOT</b>	R	000011	rD	rA	00000	0XXXX	101110
<b>NOTC</b>	R	000011	rD	rA	00000	1XXXX	101110
<b>OR</b>	R	000011	rD	rA	rB	0XXXX	101100
<b>ORC</b>	R	000011	rD	rA	rB	1XXXX	101100
<b>ORI</b>	I	101100	rD	rA	immediate		



**TABLE 2. Encoding of TA2\_SW Instruction Set**

Instruction	Format	Encoding					
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
ORIC	I	101101	rD	rA	immediate		
ORIS	I	101110	rD	rA	immediate		
PROBE	I	110010	rD	rA	offset		
RFE	R	000000	XXXXX	XXXXX	XXXXX	XXXX	111111
SLL	R	000011	rD	rA	rB	0XXXX	000000
SLLC	R	000011	rD	rA	rB	1XXXX	000000
SLLI	R	000011	rD	rA	shift_amount	0XXXX	000010
SLLIC	R	000011	rD	rA	shift_amount	1XXXX	000010
SRA	R	000011	rD	rA	rB	0XXXX	000101
SRAC	R	000011	rD	rA	rB	1XXXX	000101
SRAI	R	000011	rD	rA	shift_amount	0XXXX	000111
SRAIC	R	000011	rD	rA	shift_amount	1XXXX	000111
SRL	R	000011	rD	rA	rB	0XXXX	000001
SRLC	R	000011	rD	rA	rB	1XXXX	000001
SRLI	R	000011	rD	rA	shift_amount	0XXXX	000011
SRLIC	R	000011	rD	rA	shift_amount	1XXXX	000011
ST	I	110001	rD	rA	offset		
STBF	I	111001	rD	rA	offset		
SUB	R	000011	rD	rA	rB	0XXXX	100010
SUBC	R	000011	rD	rA	rB	1XXXX	100010
SUBE	R	000011	rD	rA	rB	0XXXX	100011
SUBEC	R	000011	rD	rA	rB	1XXXX	100011
SUBU	R	000011	rD	rA	rB	1XXXX	100100
SYS	R	000001	code				000000
TKLD	I	010010	rD	rA	offset		
TKST	I	010011	rD	rA	offset		
WADD	W	000010	wrD	wrA	wrB	0PPWW	100000
WADDC	W	000010	wrD	wrA	wrB	1PPWW	100000
WADDE	W	000010	wrD	wrA	wrB	0PPWW	100001
WADDEC	W	000010	wrD	wrA	wrB	1PPWW	100001
WAND	W	000010	wrD	wrA	wrB	0PPWW	101000
WANDC	W	000010	wrD	wrA	wrB	1PPWW	101000
WFABS	W	011101	wrD	wrA	00000	0PP10	000101
WFABSC	W	011101	wrD	wrA	00000	1PP10	000101
WFADD	W	011101	wrD	wrA	wrB	0PP10	000000
WFADDC	W	011101	wrD	wrA	wrB	1PP10	000000
WFMUL	W	011101	wrD	wrA	wrB	0PP10	000110
WFMULC	W	011101	wrD	wrA	wrB	1PP10	000110
WFNEG	W	011101	wrD	wrA	00000	0PP10	000100
WFNEGC	W	011101	wrD	wrA	00000	1PP10	000100

**TABLE 2. Encoding of TA2\_SW Instruction Set**

Instruction	Format	Encoding					
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
WFSUB	W	011101	wrD	wrA	wrB	0PP10	000001
WFSUBC	W	011101	wrD	wrA	wrB	1PP10	000001
WFTI	W	011101	wrD	wrA	00000	0PP10	000010
WFTIC	W	011101	wrD	wrA	00000	1PP10	000010
WITF	W	011101	wrD	wrA	00000	0PP10	000011
WITFC	W	011101	wrD	wrA	00000	1PP10	000011
WLD	I	110100	wrD	rA	offset		
WMRG	W	000010	wrD	wrA	wrB	CPPWW	101111
WMULES	W	000010	wrD	wrA	wrB	0PPWW	100110
WMULEU	W	000010	wrD	wrA	wrB	1PPWW	100110
WMULOS	W	000010	wrD	wrA	wrB	0PPWW	100111
WMULOU	W	000010	wrD	wrA	wrB	1PPWW	100111
WNOT	W	000010	wrD	wrA	00000	0PPWW	101110
WNOTC	W	000010	wrD	wrA	00000	1PPWW	101110
WOR	W	000010	wrD	wrA	wrB	0PPWW	101100
WORC	W	000010	wrD	wrA	wrB	1PPWW	101100
WPRM	W	000010	wrD	wrA	wrB	0PP00	001000
WPRMI	W	000010	wrD	wrA	rB	0PP00	001001
WPKS	W	000010	wrD	wrA	wrB	000WW	001110
WPKU	W	000010	wrD	wrA	wrB	100WW	001110
WSLL	W	000010	wrD	wrA	wrB	0PPWW	000000
WSLLC	W	000010	wrD	wrA	wrB	1PPWW	000000
WSLLI	W	000010	wrD	wrA	shift_amount	0PPWW	000010
WSLLIC	W	000010	wrD	wrA	shift_amount	1PPWW	000010
WSRA	W	000010	wrD	wrA	wrB	0PPWW	000101
WSRAC	W	000010	wrD	wrA	wrB	1PPWW	000101
WSRAI	W	000010	wrD	wrA	shift_amount	0PPWW	000111
WSRAIC	W	000010	wrD	wrA	shift_amount	1PPWW	000111
WSRL	W	000010	wrD	wrA	wrB	0PPWW	000001
WSRLC	W	000010	wrD	wrA	wrB	1PPWW	000001
WSRLI	W	000010	wrD	wrA	shift_amount	0PPWW	000011

**TABLE 2. Encoding of TA2\_SW Instruction Set**

Instruction	Format	Encoding					
		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
WSRLIC	W	000010	wrD	wrA	shift_amount	1PPWW	000011
WST	I	110101	wrD	rA	offset		
WSUB	W	000010	wrD	wrA	wrB	0PPWW	100010
WSUBC	W	000010	wrD	wrA	wrB	1PPWW	100010
WSUBE	W	000010	wrD	wrA	wrB	0PPWW	100011
WSUBEC	W	000010	wrD	wrA	wrB	1PPWW	100011
WSUBU	W	000010	wrD	wrA	wrB	1XXXX	100100
WUPKH	W	000010	wrD	wrA	00000	C00WW	001101
WUPKL	W	000010	wrD	wrA	00000	C00WW	001100
WXOR	W	000010	wrD	wrA	wrB	0PPWW	101010
WXORC	W	000010	wrD	wrA	wrB	1PPWW	101010
XOR	R	000011	rD	rA	rB	0XXXX	101010
XORC	R	000011	rD	rA	rB	1XXXX	101010
XORI	I	101010	rD	rA	immediate		
XORIC	I	101011	rD	rA	immediate		

**TABLE 3. Special-Purpose Registers**

NAME	SPR Number	DESCRIPTION
CC	0	LT, GT, EQ, OV, and CA bits of scalar processor
HI	1	most significant 32 bits of multiplication result, quotient of d vision
LO	2	least significant 32 bits of multiplication result, remainder of d vision
LT	8	32-bit Less Than register of WideWord Unit
GT	9	32-bit Greater Than register of WideWord Unit
EQ	10	32-bit Equal register of WideWord Unit
CA	11	32-bit Carry register of WideWord Unit
OV	12	32-bit Overfl w register of WideWord Unit
M	13	32-bit WideWord Mask register used in conditional execution
PM	14	5-bit WideWord Participation Mode register used in conditional execution
FPSR	15	32-bit WideWord Floating-Point status register

---

**TABLE 4. Protected Registers**

NAME	PR Number	DESCRIPTION
PSW	0	32-bit processor status word
SSW	1	Stored value of PSW, used in exception handling
EID	2	16-bit environment identifier register
FADR	3	32-bit address of faulting instruction (stored value of PC)
SCR <sub>0</sub> - SCR <sub>3</sub>	4 - 7	32-bit supervisor scratch registers
ESW	8	32-bit exception source word
EMR	9	32-bit exception mask register
ESR	10	32-bit exception set register
ERR	11	32-bit exception reset register
MADR	12	32-bit faulting memory address
TIMER	13	32-bit programmable delay timer
RCL	14	Low order 32 bits of real-time clock
RCH	15	High order 32 bits of real-time clock
NADR	16	32-bit address of instruction following faulting instruction (stored value of PC)

**TABLE 5. Address Translation Registers**

NAME	ATR Number	DESCRIPTION
SB <sub>0</sub> - SB <sub>7</sub>	0 - 7	32-bit local segment base registers
SL <sub>0</sub> - SL <sub>7</sub>	8 - 15	32-bit local segment limit registers
GVB <sub>0</sub> - GVB <sub>3</sub>	16 - 19	32-bit global segment virtual base registers
GL <sub>0</sub> - GL <sub>3</sub>	20 - 23	32-bit global segment limit registers
GPB <sub>0</sub> - GPB <sub>3</sub>	24 - 27	32-bit global segment physical base registers

---

## Chapter 2 - Instruction Descriptions

---

### Notation

This chapter gives detailed individual instruction descriptions. We use Big-Endian byte and bit labeling, meaning that bit/byte 0 is the most significant. Other conventions are listed in the table below.

**TABLE 6. Instruction Glossary**

Symbol	Meaning	Symbol	Meaning
$A \leftarrow B$	Assignment	MEM[EA]	Memory contents at effective address EA
$A \parallel B$	Bit string concatenation	0xvalue	Hexadecimal value
$x^y$	x replicated y times	0bvalue	Binary value
$x_{y,z}$	Selection of bits y through z from x	frX	Floating-point register X
$x \wedge y$	x bitwise ANDed with y	(rX)	Contents of general-purpose register X
$x \vee y$	x bitwise ORed with y	PC	Program counter
$x \oplus y$	x bitwise exclusive ORed with y	IADR	Instruction address
$\neg x$	bitwise inversion of x		

Note that the IADR of an instruction is equivalent to the PC value while the instruction is in the fetch stage of the pipeline.

### Precedence

The following table gives the rules of precedence and associativity for the pseudocode operators. All operators on the same line have equal precedence, and all operators on a given line have higher precedence than those on the lines below them.

**TABLE 7. Precedence of Pseudocode Operators**

Operator	Associativity
$x[n]$	left to right
$x_{y,z}$	left to right
$x^y$	left to right
$\neg$	right to left
$\times, \div$	left to right
$+, -$	left to right
$\parallel$	left to right
$=, !=, <, <=, >, >=$	left to right
$\oplus, \wedge$	left to right
$\vee$	left to right
$\leftarrow$	none

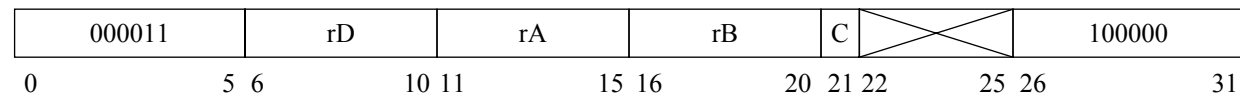
---

## addx - Add

Scalar Unit

**add**            **rD, rA, rB**    (**C = 0**)

**addc**           **rD, rA, rB**    (**C = 1**)



$$rD \leftarrow (rA) + (rB)$$

The sum (rA) + (rB) is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overfl w.

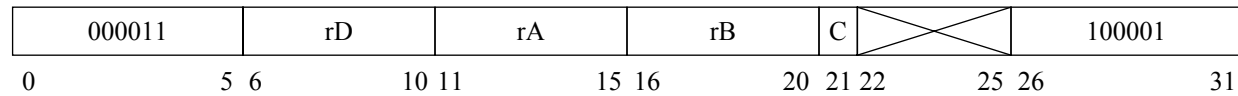
---

## addex - Add Extended

Scalar Unit

**adde**            **rD, rA, rB**     (**C = 0**)

**addec**           **rD, rA, rB**     (**C = 1**)



$$rD \leftarrow (rA) + (rB) + CA$$

The sum (rA) + (rB), using the carry bit CA as the carry in, is placed into rD.

Other registers altered:

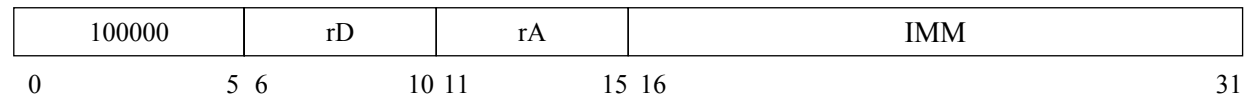
- If C =1, scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overflow.

---

## addi - Add Immediate

Scalar Unit

**addi**            **rD, rA, IMM**



$$rD \leftarrow (rA) + ((IMM_0)^{16} \parallel IMM)$$

The sum  $(rA) + IMM$  (sign-extended to form a 32-bit value) is placed into rD.

Other registers altered:

- Scalar condition code OV is set if the operation causes overflow.

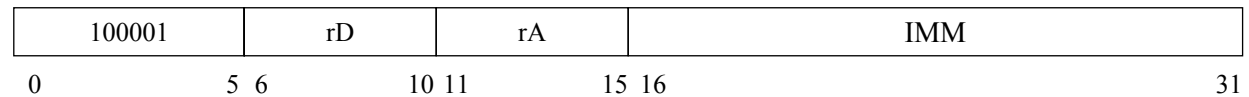


---

## addic - Add Immediate Recording Condition Code

Scalar Unit

**addic**            **rD, rA, IMM**



$$rD \leftarrow (rA) + ((IMM_0)^{16} \parallel IMM)$$

The sum  $(rA) + IMM$  (sign-extended to form a 32-bit value) is placed into rD.

Other registers altered:

- Scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overflow.

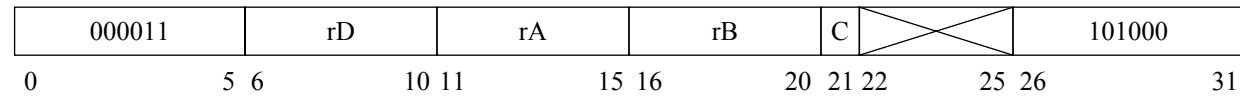
---

## andx - AND

Scalar Unit

**and**            **rD, rA, rB**    (**C = 0**)

**andc**          **rD, rA, rB**    (**C = 1**)



$$rD \leftarrow (rA) \wedge (rB)$$

The contents of rA are ANDed with rB, and the result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

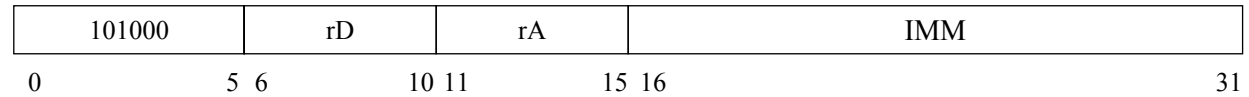
---

## andi - AND Immediate

---

Scalar Unit

**andi**            **rD, rA, IMM**



$$rD \leftarrow (rA) \wedge (0^{16} \parallel IMM)$$

The contents of rA are ANDed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

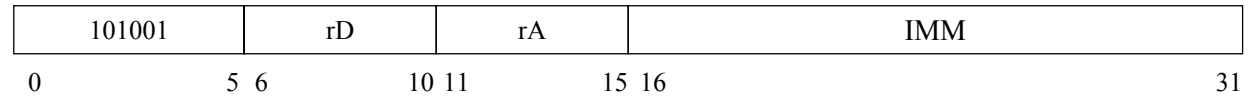
- None

---

## andic - AND Immediate Recording Condition Codes

Scalar Unit

**andic**            **rD, rA, IMM**



$$rD \leftarrow (rA) \wedge (0^{16} \parallel IMM)$$

The contents of rA are ANDed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

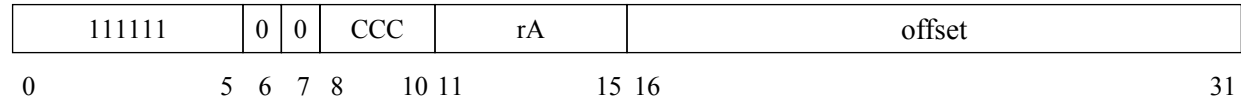
Other registers altered:

- Scalar condition code registers: LT, GT, EQ

---

## bx- Branch

### bx                      rA, offset                      (register-relative format)



### bx                      offset                      (PC-relative format)



if scalar condition indicated by CCC

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFFEC) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This branch instruction is conditional upon the scalar condition specified by CCC. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot).

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	b rA, offset	b offset
001	beq rA, offset	beq offset
010	bne rA, offset	bne offset
011	blt rA, offset	blt offset
100	ble rA, offset	ble offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
101	bgt rA, offset	bgt offset
110	bge rA, offset	bge offset
111	bov rA, offset	bov offset

Other registers altered:

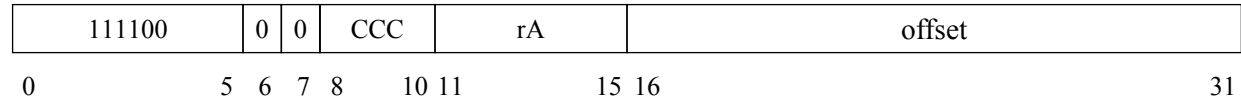
- None

The **ret** instruction is a simplified mnemonic for **b r31, 0**.

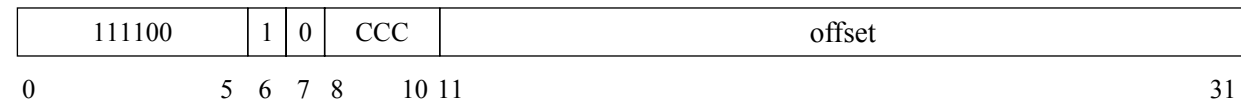
---

## bax- Branch on All

**bax**                      **rA, offset**                      **(register-relative format)**



**bax**                      **offset**                      **(PC-relative format)**



if condition indicated by CCC is true for all WideWord datapaths

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFF0) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This conditional branch instruction succeeds if the condition specified by CCC is true for all WideWord datapaths. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot).

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	b rA, offset	b offset
001	baeq rA, offset	baeq offset
010	bane rA, offset	bane offset
011	balt rA, offset	balt offset
100	bale rA, offset	bale offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
101	bagt rA, offset	bagt offset
110	bage rA, offset	bage offset
111	baov rA, offset	baov offset

Other registers altered:

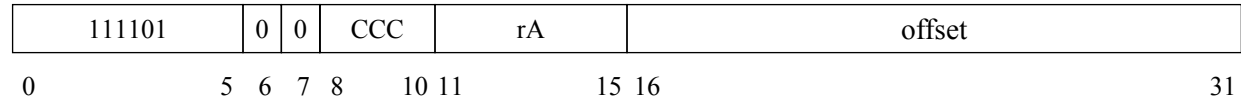
- None



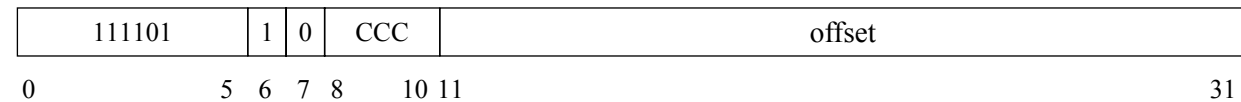
---

## bnx- Branch on None

### bnx                      rA, offset                      (register-relative format)



### bnx                      offset                      (PC-relative format)



if condition indicated by CCC is false for all WideWord datapaths

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFF0) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This conditional branch instruction succeeds if the condition specified by CCC is false for all WideWord datapaths. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot).

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	b rA, offset	b offset
001	bneq rA, offset	bneq offset
010	bnne rA, offset	bnne offset
011	bnlt rA, offset	bnlt offset
100	bnle rA, offset	bnle offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
101	bngt rA, offset	bngt offset
110	bnge rA, offset	bnge offset
111	bnov rA, offset	bnov offset

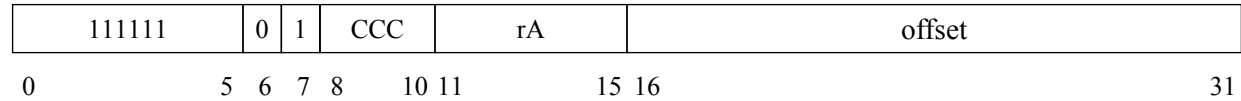
Other registers altered:

- None

---

## callx- Call

**callx                      rA, offset                      (register-relative format)**



**callx                      offset                      (PC-relative format)**



if scalar condition indicated by CCC

$$r31 \leftarrow IADR + 8$$

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFFFFFC) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This call instruction is conditional upon the scalar condition specified by CCC. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot). The effective address of the instruction following the delay slot is placed into r31.

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	call rA, offset	call offset
001	calleq rA, offset	calleq offset
010	callne rA, offset	callne offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
011	callt rA, offset	callt offset
100	callle rA, offset	callle offset
101	callgt rA, offset	callgt offset
110	callge rA, offset	callge offset
111	callov rA, offset	callov offset

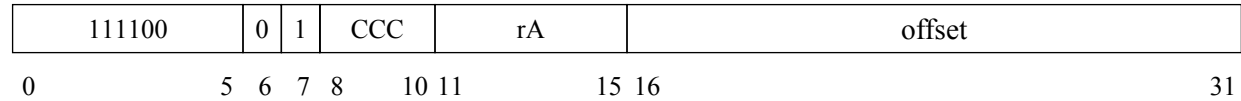
Other registers altered:

- None

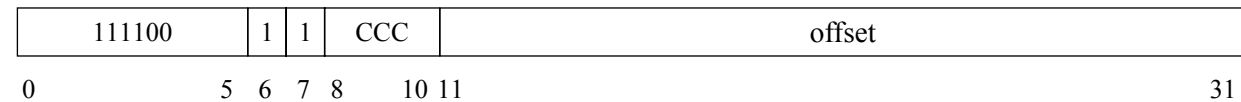
---

## callax- Call on All

### callax      rA, offset      (register-relative format)



### callax      offset      (PC-relative format)



if condition indicated by CCC is true for all WideWord datapaths

$$r31 \leftarrow IADR + 8$$

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFFFFFC) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This conditional call instruction succeeds if the condition specified by CCC is true for all WideWord datapaths. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot). The effective address of the instruction following the delay slot is placed into r31.

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	call rA, offset	call offset
001	callaeq rA, offset	callaeq offset
010	callane rA, offset	callane offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
011	callalt rA, offset	callalt offset
100	callale rA, offset	callale offset
101	callagt rA, offset	callagt offset
110	callage rA, offset	callage offset
111	callaov rA, offset	callaov offset

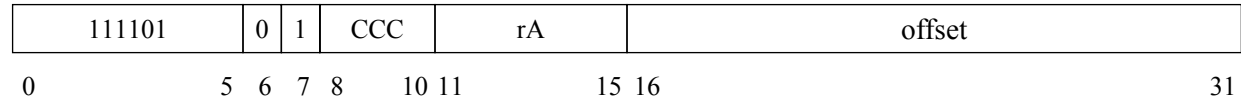
Other registers altered:

None

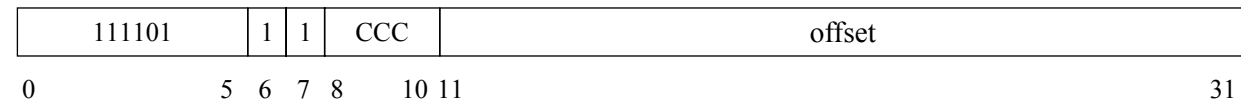
---

## callnx- Call on None

### callnx      rA, offset      (register-relative format)



### callnx      offset      (PC-relative format)



if condition indicated by CCC is false for all WideWord datapaths

$$r31 \leftarrow IADR + 8$$

if PC-relative format

$$PC \leftarrow IADR + ((offset_0)^9 \parallel offset \parallel 00)$$

else

$$PC \leftarrow ((rA) \wedge 0xFFFFFFFFC) \vee ((offset_0)^{14} \parallel offset \parallel 00)$$

This conditional call instruction succeeds if the condition specified by CCC is false for all WideWord datapaths. For the register-relative format, the target address is formed by ORing the offset with the contents of rA. For the PC-relative format, the target address is formed by adding the offset to the instruction address. In both cases, the offset is considered to be a signed instruction count, so it is shifted left two bits and sign-extended. Furthermore, the least two significant bits of the contents of rA are ignored in the register-relative format so that a proper instruction-aligned address results. The next instruction is always executed (one delay slot). The effective address of the instruction following the delay slot is placed into r31.

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
000	call rA, offset	call offset
001	callneq rA, offset	callneq offset
010	calllne rA, offset	calllne offset

---

CCC	Register-Relative Mnemonic	PC-Relative Mnemonic
011	callnlt rA, offset	callnlt offset
100	callnle rA, offset	callnle offset
101	callngt rA, offset	callngt offset
110	callnge rA, offset	callnge offset
111	callnov rA, offset	callnov offset

Other registers altered:

None



---

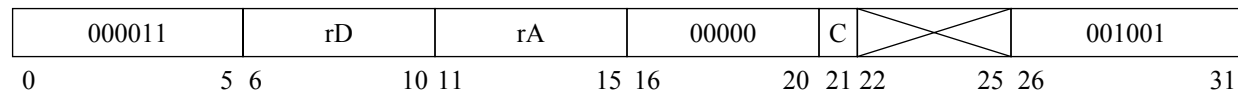
## clox - Clear Leftmost One

---

Scalar Unit

**clo**                      **rD, rA**                      (**C = 0**)

**cloc**                      **rD, rA**                      (**C = 1**)



for i = 31 to 0

    if (rA)<sub>i</sub>

*tmp* ← *i*

$rD \leftarrow (rA) \wedge (1^{tmp} \parallel 0 \parallel 1^{31-tmp})$

The contents of rA are searched to find the leftmost bit that is a one. The resulting value of clearing this bit but retaining the other bits is then stored in rD.

Other registers altered:

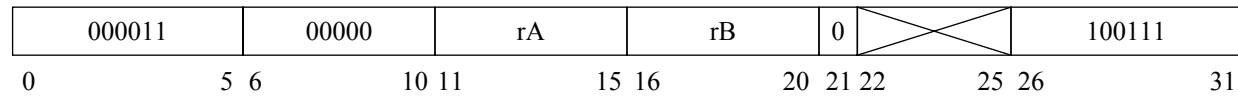
- If C =1, scalar condition code registers: LT, GT, EQ

---

## div - Divide

Scalar Unit

**div**                      **rA, rB**



$$HI \leftarrow (rA) \div (rB)$$

$$LO \leftarrow (rA) \bmod (rB)$$

The contents of rA are divided by the contents of rB, treating both operands as signed values. No condition codes are updated as a result of this operation. When the operation completes, the quotient word is loaded into special register HI, and the remainder word is loaded into special register LO. This operation requires 12 clock cycles in the worst case and thus requires some amount of scheduling.

Other registers altered:

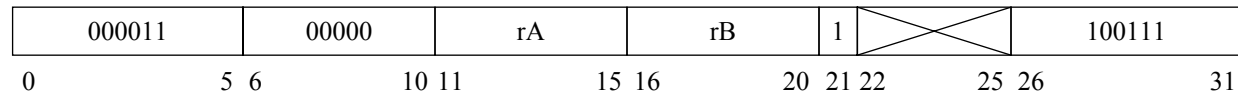
- None

---

## divu - Divide Unsigned

Scalar Unit

**divu**            **rA, rB**



$$HI \leftarrow (rA) \div (rB)$$

$$LO \leftarrow (rA) \bmod (rB)$$

The contents of rA are divided by the contents of rB, treating both operands as unsigned values. No condition codes are updated as a result of this operation. When the operation completes, the quotient word is loaded into special register HI, and the remainder word is loaded into special register LO. This operation requires 12 clock cycles in the worst case and thus requires some amount of scheduling.

Other registers altered:

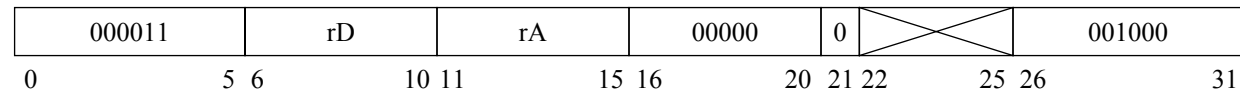
- None

---

## elo - Encode Leftmost One

Scalar Unit

**elo**                      **rD, rA**



$tmp \leftarrow 0xFFFFFFFF$

for  $i = 31$  to  $0$

    if  $(rA)_i$

$tmp \leftarrow i$

$rD \leftarrow tmp$

The contents of rA are searched to find the leftmost bit that is a one. The index of this bit is then stored in rD. If no bit of the contents of rA is a one, the value 0xFFFFFFFF is stored in rD.

Other registers altered:

- None

---

## **fabsx - Floating-Point Absolute Value**

---

Floating-Point Unit

**fabs**            **frD, frA**            (**C = 0**)

**fabsc**           **frD, frA**            (**C = 1**)

000101	frD	frA	00000	C	XXXX	000101
0	5 6	10 11	15 16	20 21 22	25 26	31

The contents of frA with bit 0, the sign bit, set to one are placed in frD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## faddx - Floating-Point Add

---

Floating-Point Unit

**fadd**            **frD, frA, frB (C = 0)**

**faddc**          **frD, frA, frB (C = 1)**

000101	frD	frA	frB	C	XXXX	000000
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (frA) + (frB)$  (using floating-point arithmetic)

Using floating point arithmetic, the sum of the single-precision floating-point contents of frA and frB is placed into frD. Floating-point exceptions may be triggered by this operation.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## **fdivx - Floating-Point Divide**

---

Floating-Point Unit

**fdiv**            **frD, frA, frB (C = 0)**

**fdivc**          **frD, frA, frB (C = 1)**

000101	frD	frA	frB	C	XXXX	000111
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (frA) \div (frB)$  (using floating-point arithmetic)

Using floating-point arithmetic, the quotient of the single-precision floating-point contents of frA and frB is placed into frD. Floating-point exceptions may be triggered by this operation.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

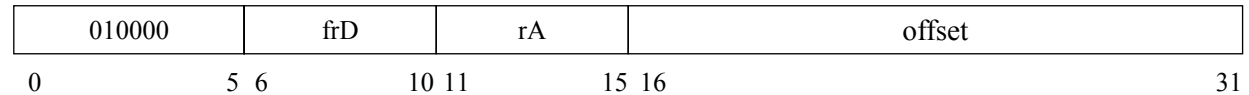
---

## **fld - Load Floating-Point Register**

---

Floating-Point Unit

**fld**                      **frD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFFC \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$frD \leftarrow MEM[EA]$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit value at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address) is then loaded into frD. If the implementation is equipped with a load buffer, this instruction loads the value from the load buffer if bits 0 through 26 of EA match the address contents of the load buffer.

Other registers altered:

- None



---

## fmulx - Floating-Point Multiply

---

Floating-Point Unit

**fmul**            **frD, frA, frB**    (**C = 0**)

**fmulc**          **frD, frA, frB** (**C = 1**)

000101	frD	frA	frB	C	XXXX	000110
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (frA) \times (frB)$  (using floating-point arithmetic)

Using floating point arithmetic, the product of the single-precision floating-point contents of frA and frB is placed into frD. Floating-point exceptions may be triggered by this operation.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## fnegx - Floating-Point Negate

---

Floating-Point Unit

**fneg**            **frD, frA**            (**C = 0**)

**fnegc**            **frD, frA**            (**C = 1**)

000101	frD	frA	00000	C	XXXX	000100
0	5 6	10 11	15 16	20 21 22	25 26	31

The contents of frA with bit 0, the sign bit, inverted are placed in frD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

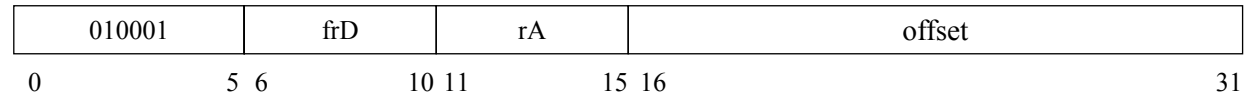
---

## **fst - Store Floating-Point Register**

---

Floating-Point Unit

**fst**                      **frD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFC \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$MEM[EA] \leftarrow frD$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit contents of frD are stored at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address). If the implementation is equipped with a store buffer, this instruction writes the value to be stored to the appropriate subfield of the store buffer, causing a flush of the prior buffer contents if bits 0 through 26 of EA do not match the address contents of the store buffer.

Other registers altered:

- None

---

## **fsubx - Floating-Point Subtract**

---

Floating-Point Unit

**fsub**            **frD, frA, frB (C = 0)**

**fsubc**          **frD, frA, frB (C = 1)**

000101	frD	frA	frB	C	XXXX	000001
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (frA) - (frB)$  (using floating-point arithmetic)

Using floating-point arithmetic, the difference of the single-precision floating-point contents of frA and frB is placed into frD. Floating-point exceptions may be triggered by this operation.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## ftix - Floating-Point to Integer

---

Floating-Point Unit

**fti**                      **frD, frA**                      (**C = 0**)

**ftic**                      **frD, frA**                      (**C = 1**)

000101	frD	frA	00000	C	XXXX	000010
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow \text{int}((frA))$  (assuming floating-point input operand)

The single-precision floating-point contents of frA are converted to a 32-bit integer, and the result is placed into frD. Floating-point exceptions may be triggered by this operation.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## itfx - Integer to Floating-Point

---

Floating-Point Unit

**itf**                      **frD, frA**                      (**C = 0**)

**itfc**                      **frD, frA**                      (**C = 1**)

000101	frD	frA	00000	C	XXXX	000011
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow fp((frA))$  (assuming integer input operand)

The integer contents of frA are converted to a 32-bit single-precision floating-point number, and the result is placed into frD. Floating-point exceptions may be triggered by this operation.

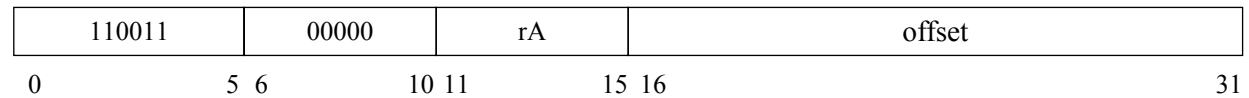
Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## icli - Instruction Cache Line Invalidate

**icli**                      **rA, offset**



$$EA \leftarrow (rA) + ((offset_0)^{16} \parallel offset)$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. If the EA is contained in the instruction cache, the cache line containing that address is invalidated.

Other registers altered:

- None

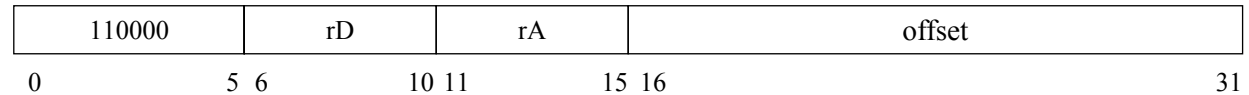
---

## ld - Load General-Purpose Register

---

Scalar Unit

**ld**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFC \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$rD \leftarrow MEM[EA]$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit word at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address) is then loaded into rD. If the implementation is equipped with a load buffer, this instruction loads the value from the load buffer if bits 0 through 26 of EA match the address contents of the load buffer.

Other registers altered:

- None



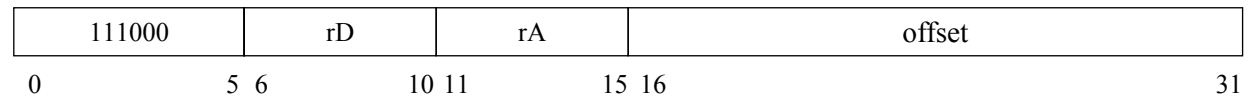
---

## ldbi - Load Buffer Invalidate

---

Scalar Unit

**ldbi**                      **rD, rA, offset**



If the implementation is equipped with a load buffer, this instruction invalidates the contents of the load buffer in the memory stage of the pipeline, which forces the next succeeding load instruction to fetch data directly from memory. The rD, rA, and offset field are ignored in the current implementation but designated for potential future use.

Other registers altered:

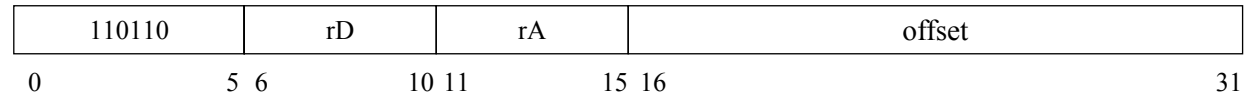
None

---

## lokl - Lock Load

Scalar Unit

**lokl**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFFC \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$rD \leftarrow MEM[EA]$$

$$LOCK \leftarrow 1$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit word at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address) is then loaded into rD. The hardware lock bit is also set and remains set until a **loks** instruction is executed or an exception occurs.

Other registers altered:

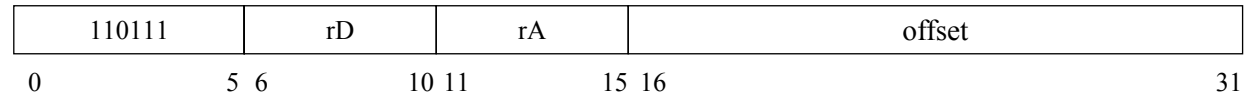
- None

---

## lks - Lock Store

Scalar Unit

**lks**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFC \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

if (LOCK = 1)

    MEM[EA]  $\leftarrow$  rD

$$rD \leftarrow LOCK^{32}$$

$$LOCK \leftarrow 0$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit word contents of rD are conditionally stored at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address). The success or failure of the store operation is indicated by the contents of rD after execution of the instruction. If an exception occurs between the last **lokl** and this **lks** instruction, the store is inhibited from taking place and the **lks** fails. The operation of **lks** is undefined when the address is different from the address used in the last **lokl**.

Other registers altered:

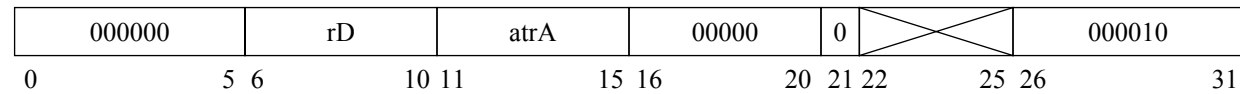
- None

---

## mfatr - Move from Address Translation Register

Scalar Unit

**mfatr**            **rD, atrA**



$rD \leftarrow (atrA)$

The contents of address translation register atrA are stored in rD. A list of the address translation registers and their encoding is found in Table 5. This instruction may be executed only in supervisor mode.

Other registers altered:

- None

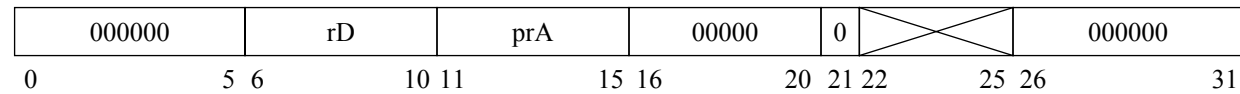
---

## mfpr - Move from Protected Register

---

Scalar Unit

**mfpr**                      **rD, prA**



$rD \leftarrow (prA)$

The contents of protected register prA are stored in rD. A list of the protected registers and their encoding is found in Table 4. This instruction may be executed only in supervisor mode.

Other registers altered:

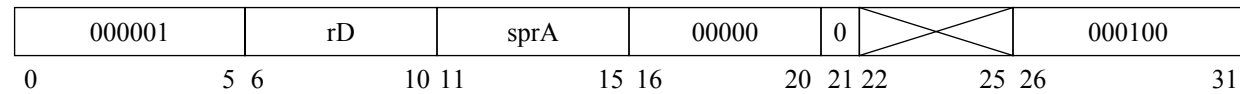
- None

---

## mfspr - Move from Special-Purpose Register

Scalar Unit

**mfspr**            **rD, sprA**



$rD \leftarrow (sprA)$

The contents of special-purpose register sprA are stored in rD. A list of the special-purpose registers and their encoding is found in Table 3.

Other registers altered:

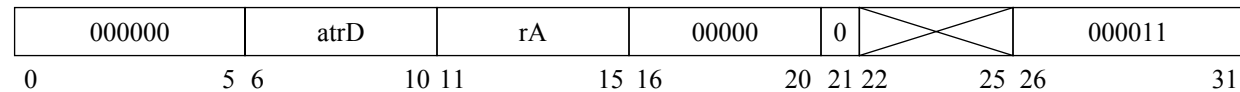
- None

---

## mtatr - Move to Address Translation Register

Scalar Unit

**mtatr**            **atrD, rA**



$atrD \leftarrow (rA)$

The contents of general-purpose register rA are stored in address translation register atrD. A list of the address translation registers and their encoding is found in Table 5. This instruction may be executed only in supervisor mode.

Other registers altered:

- None

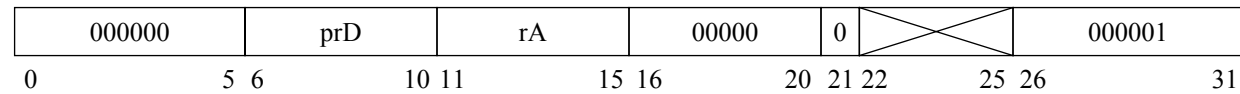
---

## mtpr - Move to Protected Register

---

Scalar Unit

**mtpr**                      **prD, rA**



$prD \leftarrow (rA)$

The contents of general-purpose register rA are stored in protected register prD. A list of the protected registers and their encoding is found in Table 4. This instruction may be executed only in supervisor mode.

Other registers altered:

- None

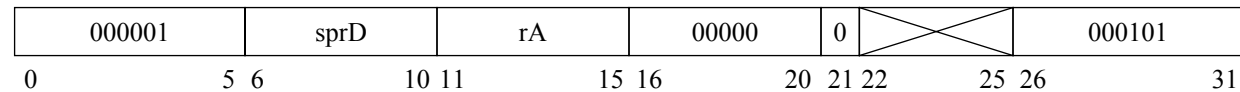


---

## mtspr - Move to Special-Purpose Register

Scalar Unit

**mtspr                  sprD, rA**



$sprD \leftarrow (rA)$

The contents of general-purpose register rA are stored in special-purpose register sprD. A list of the special-purpose registers and their encoding is found in Table 3.

Other registers altered:

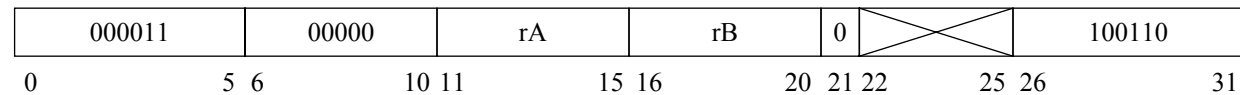
- None

---

## mul - Multiply

Scalar Unit

**mul**                      **rA, rB**



$$LO \leftarrow ((rA) \times (rB))_{32, 63}$$

$$HI \leftarrow ((rA) \times (rB))_{0, 31}$$

The contents of rA are multiplied by the contents of rB, treating both operands as signed values. No condition codes are updated as a result of this operation. When the operation completes, the low-order word of the double result is loaded into special register LO, and the high-order word is loaded into special register HI. This operation requires 4 clock cycles and thus requires some amount of scheduling.

Other registers altered:

- None

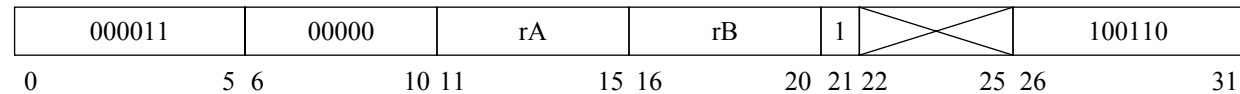
---

## mulu - Multiply Unsigned

---

Scalar Unit

**mulu**            **rA, rB**



$$LO \leftarrow ((rA) \times (rB))_{32, 63}$$

$$HI \leftarrow ((rA) \times (rB))_{0, 31}$$

The contents of rA are multiplied by the contents of rB, treating both operands as unsigned values. No condition codes are updated as a result of this operation. When the operation completes, the low-order word of the double result is loaded into special register LO, and the high-order word is loaded into special register HI. This operation requires 4 clock cycles and thus requires some amount of scheduling.

Other registers altered:

- None

---

## mvff - Move from Floating-Point to Floating-Point

---

**mvff**            **frD, frA**

000100	frD	frA	00000	X	XXXX	001010
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (frA)$

The 32-bit contents of frA are transferred to frD.

Other registers altered:

- None

---

## mvfs - Move from Floating-Point to Scalar

---

**mvfs          rD, frA**

000100	rD	frA	00000	X	XXXX	001001
0	5 6	10 11	15 16	20 21 22	25 26	31

$rD \leftarrow (frA)$

The 32-bit contents of frA are transferred to rD.

Other registers altered:

- None

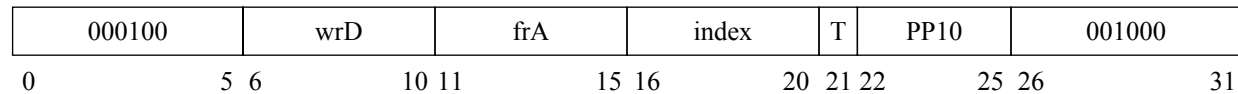
---

## mvfwx - Move from Floating-Point to WideWord

---

**mvfw**            **wrD, frA, index**            **(T = 0)**

**mvfwrp**        **wrD, frA**                        **(T = 1)**



$base \leftarrow index \wedge 0b11100$

if (T = 0)

$wrD_{base \times 8, (base \times 8) + 31} \leftarrow (frA)$

else

for i = 0 to 224 by 32

$wrD_{i, i + 31} \leftarrow (frA)$

If T=0, the contents of frA are transferred to a subfield of wrD, starting at the byte specified by the byte index. (Although a word index would be more straightforward, a byte index is used to be consistent with the mvsw instruction.) To ensure proper alignment, the least significant bits of the index are ignored. If T=1, the contents of frA are replicated to form a 256-bit value which is transferred to wrD, subject to the participation mode specified by P. The token field of wrD is undefined for this operation.

Other registers altered:

- None

---

## mvfwi - Move from Floating-Point to WideWord Indirect

---

**mvfwi**      **wrD, frA, rB**

000100	wrD	frA	rB	0	0010	101000
0	5 6	10 11	15 16	20 21 22	25 26	31

$base \leftarrow (rB)_{27,31} \wedge 0b11100$

$wrD_{base \times 8, (base \times 8) + 31} \leftarrow (frA)$

The contents of frA are transferred to a subfield of wrD, starting at the byte specified by the low-order bit contents of rB. (Although a word index would be more straightforward, a byte index is used to be consistent with the mvswi instruction.) To ensure proper alignment, the least significant bits of the index are ignored. The token field of wrD is undefined for this operation.

Other registers altered:

- None

---

## **mvsf - Move from Scalar to Floating-Point**

**mvsf          frD, rA**

000100	frD	rA	00000	X	XXXX	000110
0	5 6	10 11	15 16	20 21 22	25 26	31

$frD \leftarrow (rA)$

The 32-bit contents of rA are transferred to frD.

Other registers altered:

- None



---

## mvswx - Move from Scalar to WideWord

---

**mvsww**      **wrD, rA, index**      **(T = 0)**

**mvswrpw**   **wrD, rA**      **(T = 1)**

000100	wrD	rA	index	T	PPWW	000100
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$base \leftarrow index \wedge mask$

if (T = 0)

$wrD_{base \times 8, (base \times 8) + (size - 1)} \leftarrow (rA)_{(32 - size), 31}$

else

for i = 0 to (256 - size) by size

$wrD_{i, i + (size - 1)} \leftarrow (rA)_{(32 - size), 31}$

If T=0, some portion or all of the contents of rA are transferred to a subfield of wrD, starting at the byte specified by the byte index. Depending on the size of the data to be transferred, the least significant bits of the index may be ignored to ensure proper alignment. If T=1, the contents of rA are replicated to form a 256-bit value which is transferred to wrD, subject to the participation mode specified by PP. The token field of wrD is undefined for this operation.

Other registers altered:

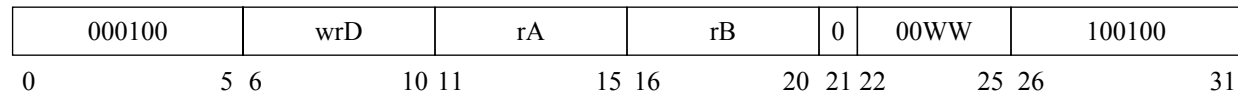
- None

---

## mvswi - Move from Scalar to WideWord Indirect

---

**mvswiw**      **wrD, rA, rB**



Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$$base \leftarrow (rB)_{27,31} \wedge mask$$

$$wrD_{base \times 8, (base \times 8) + (size - 1)} \leftarrow (rA)_{(32 - size), 31}$$

Some portion or all of the contents of rA are transferred to a subfield of wrD, starting at the byte specified by the low-order bit contents of rB. Depending on the size of the data to be transferred, the least significant bits of the contents of rB may be ignored to ensure proper alignment. The token field of wrD is undefined for this operation.

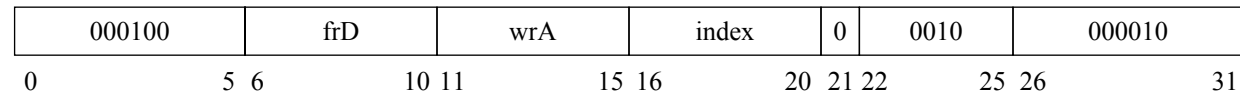
Other registers altered:

- None

---

## mvwf - Move from WideWord to Floating-Point

**mvwf**      **frD, wrA, index**



$base \leftarrow index \wedge 0b11100$

$frD \leftarrow (wrA)_{base \times 8, (base \times 8) + 31}$

A 32-bit subfield of the contents of wrA starting at the byte specified by the byte index are transferred to frD. (Although a word index would be more straightforward, a byte index is used to be consistent with the mvws instruction.) The least significant bits of the index are ignored to ensure proper alignment. The token field of wrA is ignored in this operation.

Other registers altered:

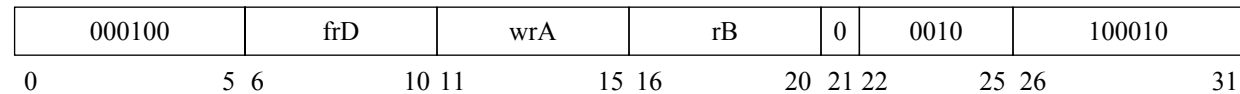
- None

---

## mvwfi - Move from WideWord to Floating-Point Indirect

---

**mvwfi**      **frD, wrA, rB**



$$base \leftarrow (rB)_{27,31} \wedge 0b11100$$

$$frD \leftarrow (wrA)_{base \times 8, (base \times 8) + 31}$$

A 32-bit subfield of the contents of wrA starting at the byte specified by the low-order bits of the contents of rB are transferred to frD. (Although a word index would be more straightforward, a byte index is used to be consistent with the mvwsi instruction.) The least significant bits of the index are ignored to ensure proper alignment. The token field of wrA is ignored in this operation

Other registers altered:

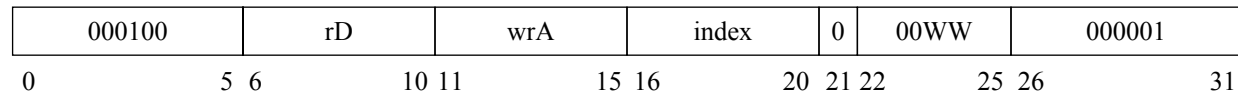
- None

---

## mvws - Move from WideWord to Scalar

---

**mvws<sub>w</sub>      rD, wrA, index**



Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$$base \leftarrow index \wedge mask$$

$$rD_{(32-size), 31} \leftarrow (wrA)_{base \times 8, (base \times 8) + (size - 1)}$$

if (size != 32)

$$rD_{0, (32-size-1)} \leftarrow 0^{(32-size)}$$

A subfield of the contents of wrA starting at the byte specified by the byte index are transferred to rD. Depending on the size of the data to be transferred, the least significant bits of the index may be ignored to ensure proper alignment. For data sizes less than 32 bits, the high-order bits of rD are cleared. The token field of wrA is ignored in this operation

Other registers altered:

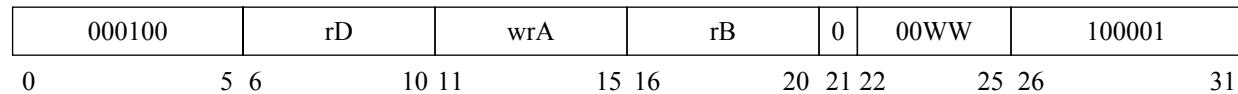
- None

---

## mvwsi - Move from WideWord to Scalar Indirect

---

**mvwsiw**      **rD, wrA, rB**



Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$$base \leftarrow (rB)_{27,31} \wedge mask$$

$$rD_{(32-size),31} \leftarrow (wrA)_{base \times 8, (base \times 8) + (size - 1)}$$

if (size != 32)

$$rD_{0, (32-size-1)} \leftarrow 0^{(32-size)}$$

A subfield of the contents of wrA starting at the byte specified by the low-order bits of the contents of rB are transferred to rD. Depending on the size of the data to be transferred, the least significant bits of the contents of rB may be ignored to ensure proper alignment. For data sizes less than 32 bits, the high-order bits of rD are cleared. The token field of wrA is ignored in this operation.

Other registers altered:

- None

---

## mvwwx - Move from WideWord to WideWord

---

**mvwwp**      **wrD, wrA**      **(T = 0)**

**mvwwrpw** **wrD, wrA, index**      **(T = 1)**

000100	wrD	wrA	index	T	PPWW	000000
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$base \leftarrow index \wedge mask$

if (T = 0)

$wrD \leftarrow (wrA)$

else

    for i = 0 to (256 - size) by size

$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{base \times 8, (base \times 8) + (size - 1)}$

If T=0, the entire 256-bit contents of wrA are transferred to wrD, subject to the participation mode specified by PP. If T=1, the subfield of wrA starting at the byte specified by the byte index and of the size indicated by the WW bits is replicated to form a 256-bit value which is transferred to wrD, subject to the participation mode specified by PP. Depending on the size of the data to be transferred, the least significant bits of the index may be ignored to ensure proper alignment. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- None

---

## mvwwir - Move from WideWord to WideWord Indirect Replicating

---

### mvwwirpw wrD, wrA, rB

000100	wrD	wrA	rB	1	PPWW	100000
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	mask
00	8	0b11111
01	16	0b11110
10	32	0b11100

$$base \leftarrow (rB)_{27,31} \wedge mask$$

for i = 0 to (256 - size) by size

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{base \times 8, (base \times 8) + (size - 1)}$$

The subfield of wrA starting at the byte specified by the low-order bits of the contents of rB and of the size indicated by the WW bits is replicated to form a 256-bit value which is transferred to wrD, subject to the participation mode specified by PP. Depending on the size of the data to be transferred, the least significant bits of the contents of rB may be ignored to ensure proper alignment. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- None



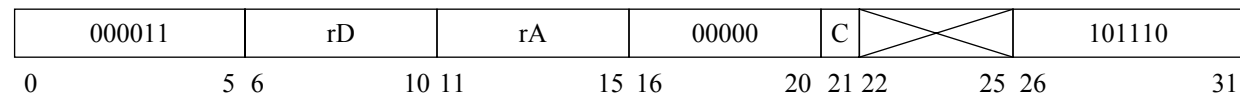
---

## notx - NOT

Scalar Unit

**not**            **rD, rA**            (**C = 0**)

**notc**           **rD, rA**            (**C = 1**)



$$rD \leftarrow \neg(rA)$$

The bitwise inversion of the contents of rA is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

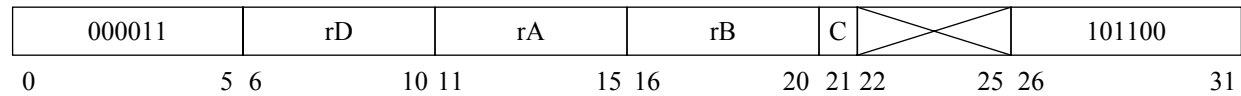
---

## orx - OR

Scalar Unit

**or**                **rD, rA, rB**     (**C = 0**)

**orc**              **rD, rA, rB**     (**C = 1**)



$$rD \leftarrow (rA) \vee (rB)$$

The contents of rA are ORed with rB, and the result is placed into rD.

Other registers altered:

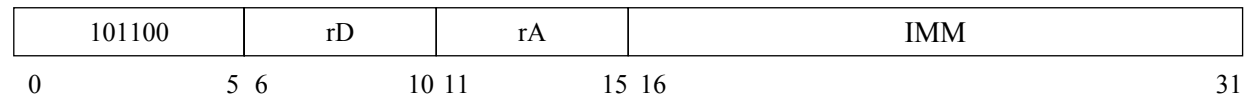
- If C =1, scalar condition code registers: LT, GT, EQ

---

## ori - OR Immediate

Scalar Unit

**ori**                      **rD, rA, IMM**



$$rD \leftarrow (rA) \vee (0^{16} \parallel IMM)$$

The contents of rA are ORed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

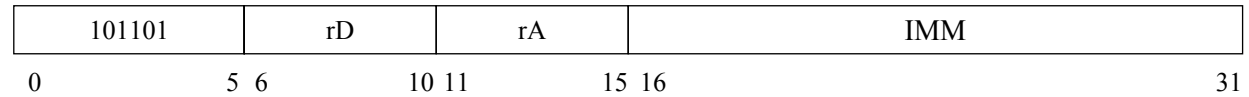
- None

---

## oric - OR Immediate Recording Condition Codes

Scalar Unit

**oric**                      **rD, rA, IMM**



$$rD \leftarrow (rA) \vee (0^{16} \parallel IMM)$$

The contents of rA are ORed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

- Scalar condition code registers: LT, GT, EQ

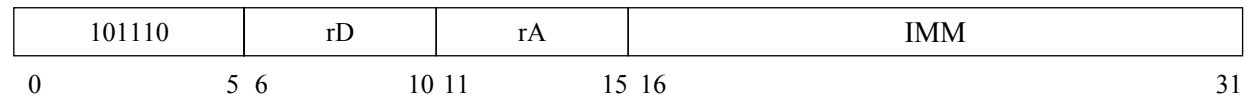
---

## oris - OR Immediate Shifted

---

Scalar Unit

**oris**                      **rD, rA, IMM**



$$rD \leftarrow (rA) \vee (IMM \parallel 0^{16})$$

The contents of rA are ORed with IMM (appended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

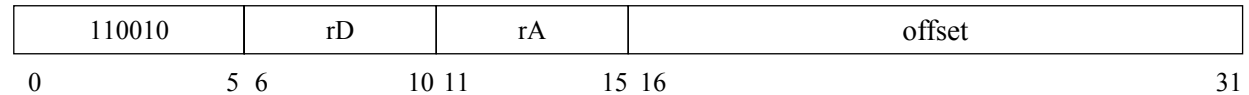
- None

---

## probe - Probe Address

Scalar Unit

**probe**            **rD, rA, offset**



$EA \leftarrow (rA) + ((offset_0)^{16} \parallel offset)$

if EA is locally mapped

$rD \leftarrow 0xFFFFFFFF$

else

$rD \leftarrow 0x00000000$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The effective address is then forwarded to the address translation hardware to determine if the address is a valid local address. The success or failure of the operation is indicated by the contents of rD after execution of the instruction.

Other registers altered:

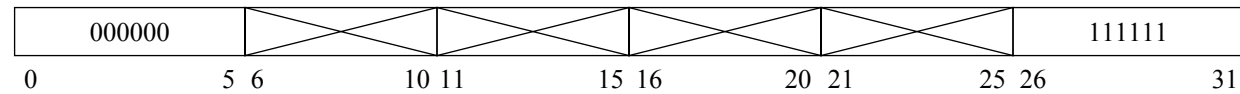
- None

---

## rfe - Return from Exception

---

### rfe



$PC \leftarrow (FADR)$

$PSW \leftarrow (SSW)$

The program counter, PC, is loaded with the contents of the protected register FADR. Similarly, the PSW is loaded with the contents of SSW. The next instruction is always executed (one delay slot).

Other registers altered:

- None

---

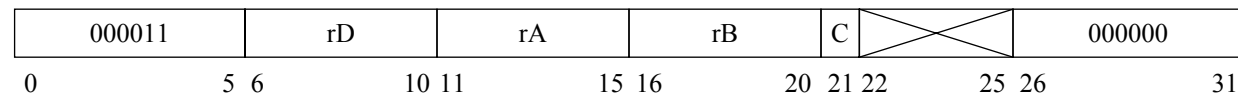
## sllx - Shift Left Logical

---

Scalar Unit

**sll**                    **rD, rA, rB**     (**C = 0**)

**sllc**                   **rD, rA, rB**     (**C = 1**)



$$s \leftarrow (rB)_{27,31}$$

$$rD \leftarrow (rA)_{s,31} \parallel 0^s$$

The contents of rA are shifted left by the number of bits specified by the low order five bits contained as contents of rB, inserting zeros into the low order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ



---

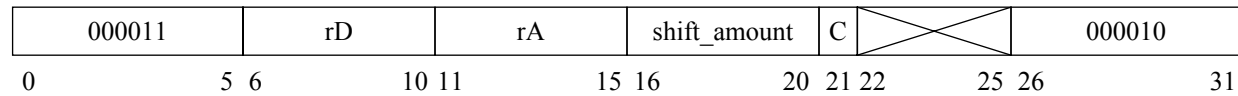
## sllix - Shift Left Logical Immediate

---

Scalar Unit

**slli**            **rD, rA, shift\_amount**    (**C = 0**)

**sllic**           **rD, rA, shift\_amount**    (**C = 1**)



$s \leftarrow \text{shift\_amount}$

$rD \leftarrow (rA)_{s, 31} \parallel 0^s$

The contents of rA are shifted left by *shift\_amount* bits, inserting zeros into the low-order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

---

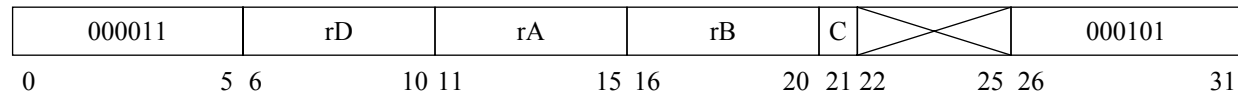
## srax - Shift Right Arithmetic

---

Scalar Unit

**sra**            **rD, rA, rB**    (**C = 0**)

**srac**           **rD, rA, rB**    (**C = 1**)



$$s \leftarrow (rB)_{27,31}$$

$$rD \leftarrow ((rA)_0)^s \parallel (rA)_{0,(31-s)}$$

The contents of rA are shifted right by the number of bits specified by the low order five bits contained as contents of rB, sign-extending the high-order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

---

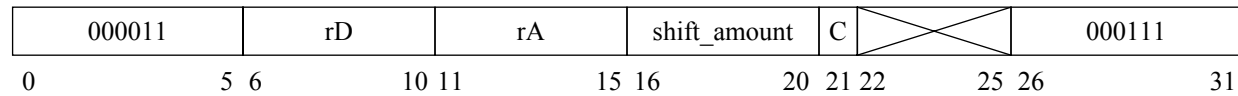
## sraix - Shift Right Arithmetic Immediate

---

Scalar Unit

**srai**            **rD, rA, shift\_amount**    (**C = 0**)

**sraic**           **rD, rA, shift\_amount**    (**C = 1**)



$s \leftarrow \text{shift\_amount}$

$rD \leftarrow ((rA)_0)^s \parallel (rA)_{0, (31-s)}$

The contents of rA are shifted right by *shift\_amount* bits, sign-extending the high-order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

---

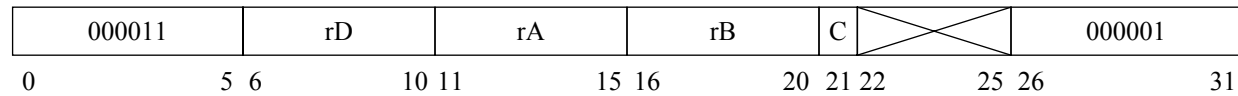
## srlx - Shift Right Logical

---

Scalar Unit

**srl**                    **rD, rA, rB**     (**C = 0**)

**srlc**                   **rD, rA, rB**     (**C = 1**)



$$s \leftarrow (rB)_{27,31}$$

$$rD \leftarrow 0^s \parallel (rA)_{0,(31-s)}$$

The contents of rA are shifted right by the number of bits specified by the low order five bits contained as contents of rB, inserting zeros into the high-order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

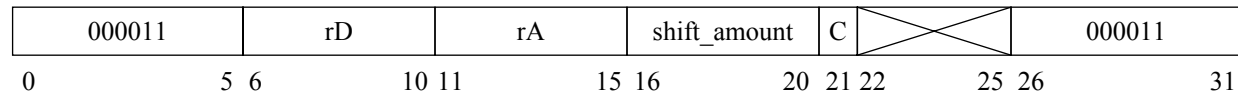
---

## srlix - Shift Right Logical Immediate

Scalar Unit

**srli**            **rD, rA, shift\_amount**    (**C = 0**)

**srlic**           **rD, rA, shift\_amount**    (**C = 1**)



$s \leftarrow \text{shift\_amount}$

$$rD \leftarrow 0^s \parallel (rA)_{0, (31-s)}$$

The contents of rA are shifted right by *shift\_amount* bits, inserting zeros into the high-order bits of the result. The result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ

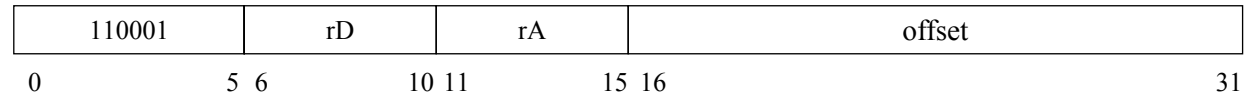
---

## st - Store General-Purpose Register

---

Scalar Unit

**st**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFC \wedge ((rA) + (offset_0)^{16} \parallel offset)$$

$$MEM[EA] \leftarrow rD$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 32-bit word contents of rD are stored at the memory location specified by EA (ignoring the least two significant bits to ensure a 32-bit aligned address). If the implementation is equipped with a store buffer, this instruction writes the value to be stored to the appropriate subfield of the store buffer, causing a flush of the prior buffer contents if bits 0 through 26 of EA do not match the address contents of the store buffer.

Other registers altered:

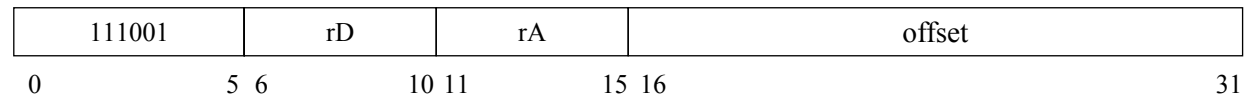
- None

---

## stbf - Store Buffer Flush

Scalar Unit

**stbf**                      **rD, rA, offset**



If the implementation is equipped with a store buffer, this instruction forces a flush of the store buffer to memory. The rD, rA, and offset fields are ignored in the current implementation but designated for potential future use.

Other registers altered:

- None

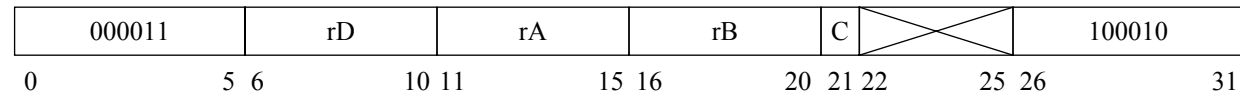
---

## subx - Subtract

Scalar Unit

**sub**            **rD, rA, rB**    (**C = 0**)

**subc**           **rD, rA, rB**    (**C = 1**)



$$rD \leftarrow (rA) + \neg(rB) + 1$$

The contents of rB are subtracted from the contents of rA, and the result is placed into rD.

Other registers altered:

- If C =1, scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overflow.



---

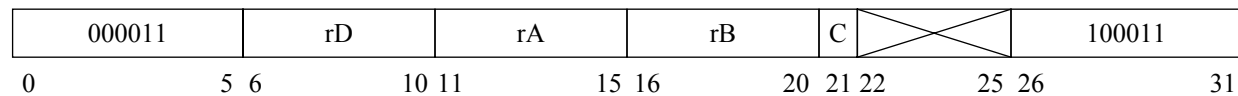
## subex - Subtract Extended

---

Scalar Unit

**sube**            **rD, rA, rB**    (**C = 0**)

**subec**           **rD, rA, rB**    (**C = 1**)



$$rD \leftarrow (rA) + \neg(rB) + CA$$

The contents of rB are subtracted from the contents of rA, using the carry bit CA as the carry in, and the result is placed into rD.

Other registers altered:

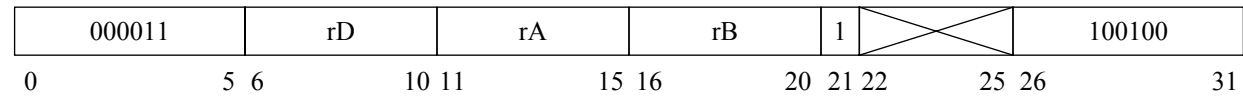
- If C =1, scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overflow.

---

## subu - Subtract

Scalar Unit

**subu**            **rD, rA, rB**



$$rD \leftarrow (rA) + \neg(rB) + 1$$

The contents of rB are subtracted from the contents of rA, and the result is placed into rD. This instruction is identical to **sub** except that the OV condition code is updated to reflect unsigned arithmetic

Other registers altered:

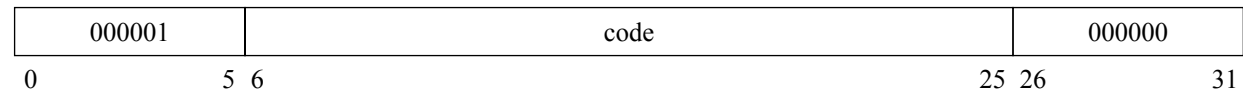
- Scalar condition code registers: LT, GT, EQ, CA
- Scalar condition code OV is set if the operation causes overflow.

---

## sys - System Call

---

### sys



A system call is made by setting bit 19 of the ESW (Exception Source Word) register which in turn triggers an exception. Refer to the TA2\_SW RISC Processor Architecture manual for details regarding exceptions.

Other registers altered:

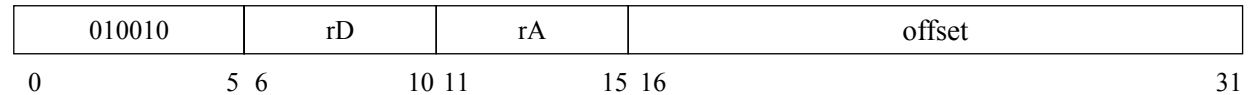
- None

---

## tkld - Load Token Field into General-Purpose Register

Scalar Unit

**tkld**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFFE0 \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$rD_{16,31} \leftarrow \text{token field of MEM}[EA]$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 16-bit token field associated with the 256-bit wide word at the memory location specified by EA (ignoring the least five significant bits to ensure a 256-bit aligned address) is then loaded into the least significant half of rD.

Other registers altered:

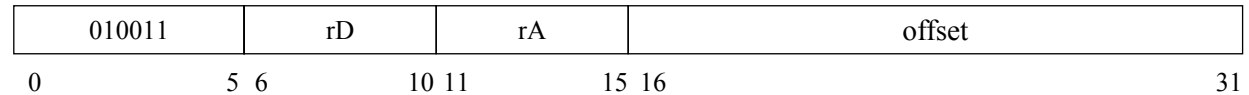
- None

---

## tkst - Store General-Purpose Register into Token Field

Scalar Unit

**tkst**                      **rD, rA, offset**



$$EA \leftarrow 0xFFFFFFFF0 \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

token field of MEM[EA]  $\leftarrow rD_{16, 31}$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The least significant half of rD is then stored into the 16-bit token field associated with the 256-bit wide word at the memory location specified by EA (ignoring the least five significant bits to ensure a 256-bit aligned address)

Other registers altered:

- None

---

## waddx - WideWord Add

---

WideWord Unit

**waddpw**     **wrD, wrA, wrB (C = 0)**

**waddecpw**   **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	100000
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} + (wrB)_{i, i + (size - 1)}$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate sums of the aligned data field of wrA and wrB are placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ, CA
- A WideWord OV condition code bit is set if the operation in its corresponding datapath causes overflow.

---

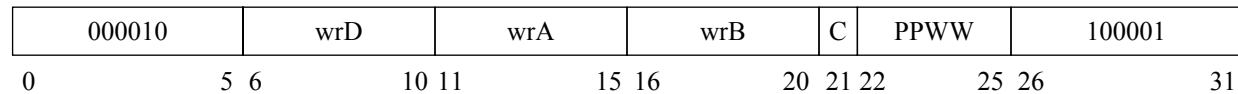
## waddex - WideWord Add Extended

---

WideWord Unit

**waddepw** wrD, wrA, wrB (C = 0)

**waddecpw** wrD, wrA, wrB (C = 1)



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} + (wrB)_{i, i + (size - 1)} + CA_{i/8}$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate sums of the aligned data field of wrA and wrB are placed into wrD, subject to participation. Each data field uses the associated bit of the WideWord Carry register as a carry in for the operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ, CA
- A WideWord OV condition code bit is set if the operation in its corresponding datapath causes overflow.

---

## wandx - WideWord AND

---

WideWord Unit

**wandpw**     **wrD, wrA, wrB (C = 0)**

**wandcpw**    **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	101000
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} \wedge (wrB)_{i, i + (size - 1)}$$

The 256-bit contents of wrA are ANDed with the 256-bit contents of wrB, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies and how condition codes are updated for this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ



---

## wfabsx - WideWord Floating-Point Absolute Value

---

WideWord Unit

**wfabs $p$**       **wrD, wrA**      (**C = 0**)

**wfabs $c$ p**      **wrD, wrA**      (**C = 1**)

011101	wrD	wrA	00000	C	PP10	000101
0	5 6	10 11	15 16	20 21 22	25 26	31

for  $i = 0$  to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i, i+31} \leftarrow 1 \parallel (wrA)_{i+1, i+31}$$

The 256-bit contents of wrA are treated as 8 single-precision floating-point operands. For each operand in wrA, the operand with bit 0, the sign bit, set to one is placed into the corresponding field of wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## wfaddx - WideWord Floating-Point Add

---

WideWord Unit

**wfaddp**      **wrD, wrA, wrB (C = 0)**

**wfaddec**    **wrD, wrA, wrB (C = 1)**

011101	wrD	wrA	wrB	C	PP10	000000
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i,i+31} \leftarrow (wrA)_{i,i+31} + (wrB)_{i,i+31} \text{ (using floating-point arithmetic)}$$

The 256-bit contents of wrA and wrB are treated as 8 single-precision floating-point operands. The aggregate floating-point sums of the aligned data field of wrA and wrB are placed into wrD, subject to participation. Floating-point exceptions may be triggered by this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## wfmulx - WideWord Floating-Point Multiply

---

WideWord Unit

**wfmulp**      **wrD, wrA, wrB (C = 0)**

**wfmulcp**    **wrD, wrA, wrB (C = 1)**

011101	wrD	wrA	wrB	C	PP10	000110
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i,i+31} \leftarrow (wrA)_{i,i+31} \times (wrB)_{i,i+31} \text{ (using floating-point arithmetic)}$$

The 256-bit contents of wrA and wrB are treated as 8 single-precision floating-point operands. The aggregate floating-point products of the aligned data field of wrA and wrB are placed into wrD, subject to participation. Floating-point exceptions may be triggered by this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## wfnegx - WideWord Floating-Point Negate

---

WideWord Unit

**wfnegp**      **wrD, wrA**      (**C = 0**)

**wfnegcp**    **wrD, wrA**      (**C = 1**)

011101	wrD	wrA	00000	C	PP10	000100
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i,i+31} \leftarrow \neg(wrA)_i \parallel (wrA)_{i+1,i+31}$$

The 256-bit contents of wrA are treated as 8 single-precision floating-point operands. For each operand in wrA, the operand with bit 0, the sign bit, inverted is placed into the corresponding field of wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## wfsubx - WideWord Floating-Point Subtract

---

WideWord Unit

**wfsubp**      **wrD, wrA, wrB (C = 0)**

**wfsubcp**     **wrD, wrA, wrB (C = 1)**

011101	wrD	wrA	wrB	C	PP10	000001
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i,i+31} \leftarrow (wrA)_{i,i+31} - (wrB)_{i,i+31} \text{ (using floating-point arithmetic)}$$

The 256-bit contents of wrA and wrB are treated as 8 single-precision floating-point operands. The aggregate floating-point differences of the aligned data field of wrA and wrB are placed into wrD, subject to participation. Floating-point exceptions may be triggered by this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## wftix - WideWord Floating-Point to Integer

---

WideWord Unit

**wftip**            **wrD, wrA**        (**C = 0**)

**wfticp**           **wrD, wrA**        (**C = 1**)

011101	wrD	wrA	00000	C	PP10	000010
0	5 6	10 11	15 16	20 21 22	25 26	31

for  $i = 0$  to 224 by 32

if PP bits and conditions are set accordingly

$wrD_{i, i+31} \leftarrow \text{int}((wrA)_{i, i+31})$  (assuming floating-point input operand

The 256-bit contents of wrA are treated as 8 single-precision floating-point operands. Each single-precision floating-point operand is converted to a 32-bit integer, and the aggregation of these 8 integers are placed into wrD, subject to participation. Floating-point exceptions may be triggered by this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

---

## witfx - WideWord Integer to Floating-Point

---

WideWord Unit

**witfp**            **wrD, wrA**        (**C = 0**)

**witfcp**           **wrD, wrA**        (**C = 1**)

011101	wrD	wrA	00000	C	PP10	000011
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 224 by 32

if PP bits and conditions are set accordingly

$$wrD_{i,i+31} \leftarrow fp((wrA)_{i,i+31}) \text{ (assuming integer input operand)}$$

The 256-bit contents of wrA are treated as eight 32-bit integer operands. Each integer operand is converted to a single-precision floating-point number, and the aggregation of these 8 single-precision floating-point numbers are placed into wrD, subject to participation. Floating-point exceptions may be triggered by this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ
- FPSR may also be updated if any floating-point exceptions occur.

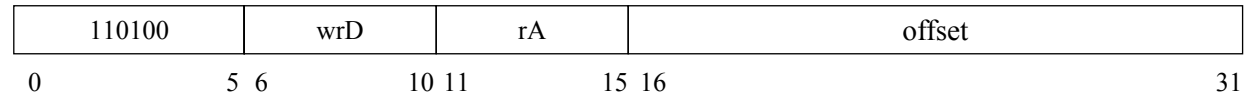
---

## wld - Load WideWord Register

---

WideWord Unit

**wld**                      **wrD, rA, offset**



$$EA \leftarrow 0xFFFFFFFF0 \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$wrD \leftarrow MEM[EA]$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 256-bit data value and 16-bit token value at the memory location specified by EA (ignoring the least five significant bits to ensure a 256-bit aligned address) are then loaded into wrD.

Other registers altered:

- None



---

## wmrgx - WideWord Merge

---

WideWord Unit

**wmrgcp**     **wrD, wrA, wrB (C = 0)**

**wmrgccp**    **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	101111
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	CC	Mnemonic (c)
00	EQ	eq
01	LT	lt
10	GT	gt
11	M	m

for i = 0 to 248 by 8

if PP bits and conditions are set accordingly

if  $CC_{i/8} = 1$

$$wrD_{i,i+7} \leftarrow (wrA)_{i,i+7}$$

else

$$wrD_{i,i+7} \leftarrow (wrB)_{i,i+7}$$

Each bit of the WideWord condition code register specified by the WW bits of the instruction serves as a selector. If the bit is 1, the corresponding byte contents of wrA are placed into the corresponding byte lane of wrD, subject to participation. If the bit is 0, the corresponding byte contents of wrB are placed into the corresponding byte lane of wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

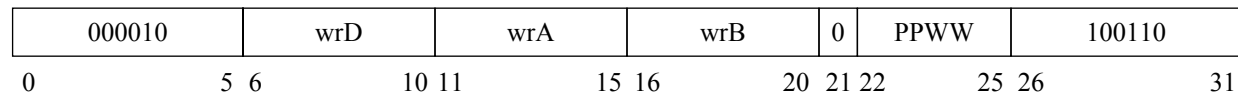
If C = 1, WideWord condition code registers: LT, GT, EQ

---

## wmules - WideWord Multiply Even Signed

WideWord Unit

**wmules<sub>pw</sub> wrD, wrA, wrB**



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - 2 \times \text{size})$  by  $2 \times \text{size}$

if PP bits and conditions are set accordingly

$$wrD_{i, i + (2 \times \text{size} - 1)} \leftarrow (wrA)_{i, i + (\text{size} - 1)} \times (wrB)_{i, i + (\text{size} - 1)}$$

Each even-numbered signed-integer byte or half-word of wrA is multiplied by the corresponding signed-integer byte or half-word of wrB, where the WW field determines if the 256-bit contents of wrA and wrB are treated as bytes or half-words. The resulting signed halfword or word products are placed, in the same order, into wrD, subject to participation. No condition codes are updated as a result of this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

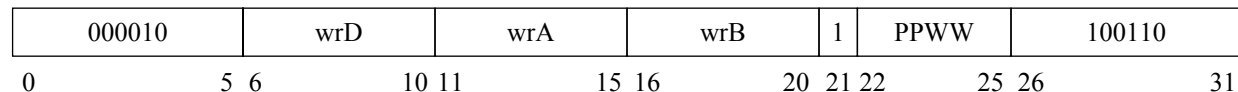
- None

---

## wmuleu - WideWord Multiply Even Unsigned

WideWord Unit

**wmuleupw wrD, wrA, wrB**



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - 2 \times \text{size})$  by  $2 \times \text{size}$

if PP bits and conditions are set accordingly

$$wrD_{i, i + (2 \times \text{size} - 1)} \leftarrow (wrA)_{i, i + (\text{size} - 1)} \times (wrB)_{i, i + (\text{size} - 1)}$$

Each even-numbered unsigned-integer byte or half-word of wrA is multiplied by the corresponding unsigned-integer byte or half-word of wrB, where the WW field determines if the 256-bit contents of wrA and wrB are treated as bytes or half-words. The resulting unsigned half-word or word products are placed, in the same order, into wrD, subject to participation. No condition codes are updated as a result of this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

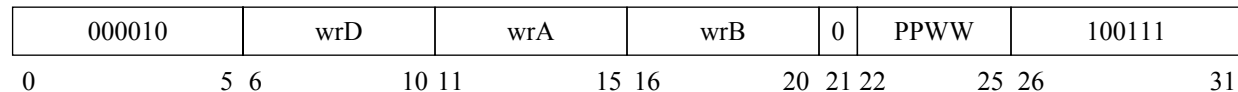
- None

---

## wmulos - WideWord Multiply Odd Signed

WideWord Unit

**wmulos<sub>pw</sub> wrD, wrA, wrB**



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - 2 \times \text{size})$  by  $2 \times \text{size}$

if PP bits and conditions are set accordingly

$$wrD_{i, i + (2 \times \text{size} - 1)} \leftarrow (wrA)_{i + \text{size}, i + (2 \times \text{size} - 1)} \times (wrB)_{i + \text{size}, i + (2 \times \text{size} - 1)}$$

Each odd-numbered signed-integer byte or half-word of wrA is multiplied by the corresponding signed-integer byte or half-word of wrB, where the WW field determines if the 256-bit contents of wrA and wrB are treated as bytes or half-words. The resulting signed halfword or word products are placed, in the same order, into wrD, subject to participation. No condition codes are updated as a result of this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

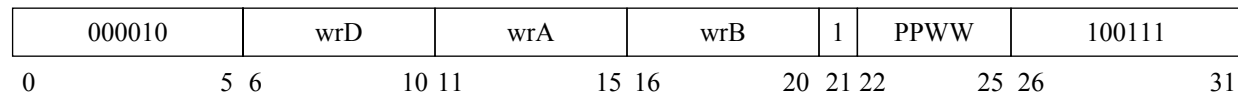
- None

---

## wmulou - WideWord Multiply Odd Unsigned

WideWord Unit

### wmulou $_{pw}$ wrD, wrA, wrB



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - 2 \times \text{size})$  by  $2 \times \text{size}$

if PP bits and conditions are set accordingly

$$wrD_{i, i + (2 \times \text{size} - 1)} \leftarrow (wrA)_{i + \text{size}, i + (2 \times \text{size} - 1)} \times (wrB)_{i + \text{size}, i + (2 \times \text{size} - 1)}$$

Each odd-numbered unsigned-integer byte or half-word of wrA is multiplied by the corresponding unsigned-integer byte or half-word of wrB, where the WW field determines if the 256-bit contents of wrA and wrB are treated as bytes or half-words. The resulting unsigned half-word or word products are placed, in the same order, into wrD, subject to participation. No condition codes are updated as a result of this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- None

---

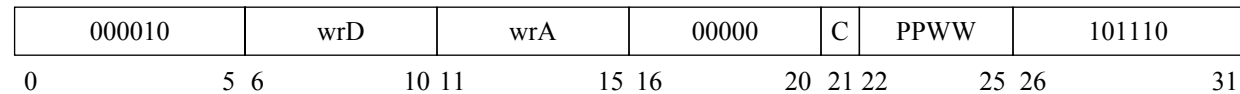
## wnotx - WideWord NOT

---

WideWord Unit

**wnotpw**      **wrD, wrA**      (**C = 0**)

**wnotcpw**    **wrD, wrA**      (**C = 1**)



Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow \neg(wrA)_{i, i + (size - 1)}$$

The 256-bit contents of wrA are bitwise inverted, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies and how condition codes are updated for this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C=1, WideWord condition code registers: LT, GT, EQ

---

## worx - WideWord OR

---

WideWord Unit

**worpw**      **wrD, wrA, wrB (C = 0)**

**worc pw**      **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	101100
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} \vee (wrB)_{i, i + (size - 1)}$$

The 256-bit contents of wrA are ORed with the 256-bit contents of wrB, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies and how condition codes are updated for this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ

---

## wpk<sub>sx</sub> - WideWord Pack Signed

---

WideWord Unit

**wpk<sub>sw</sub>**      **wrD, wrA, wrB**

000010	wrD	wrA	wrB	0	00WW	001110
0	5 6	10 11	15 16	20 21 22	26 27	31

Variable values in the following equations are as follows:

WW Value	size	min	max
01	16	$-2^7$	$2^7 - 1$
10	32	$-2^{15}$	$2^{15} - 1$

for  $i = 0$  to  $(128 - (\text{size}/2))$  by  $(\text{size}/2)$

if  $(wrA)_{i \times 2, (i \times 2) + \text{size} - 1} < \text{min}$

$wrD_{i, i + (\text{size}/2) - 1} \leftarrow \text{min}$

else if  $(wrA)_{i \times 2, (i \times 2) + \text{size} - 1} > \text{max}$

$wrD_{i, i + (\text{size}/2) - 1} \leftarrow \text{max}$

else

$wrD_{i, i + (\text{size}/2) - 1} \leftarrow (wrA)_{(i \times 2) + (\text{size}/2), (i \times 2) + \text{size} - 1}$

if  $(wrB)_{i \times 2, (i \times 2) + \text{size} - 1} < \text{min}$

$wrD_{128 + i, 128 + i + (\text{size}/2) - 1} \leftarrow \text{min}$

else if  $(wrB)_{i \times 2, (i \times 2) + \text{size} - 1} > \text{max}$

$wrD_{128 + i, 128 + i + (\text{size}/2) - 1} \leftarrow \text{max}$

else

$wrD_{128 + i, 128 + i + (\text{size}/2) - 1} \leftarrow (wrB)_{(i \times 2) + (\text{size}/2), (i \times 2) + \text{size} - 1}$



---

Let the source vector be the concatenation of the contents of wrA followed by wrB. Each signed integer half-word or word, as specified by the WW bits, of the source vector is converted to a signed integer byte or half-word, respectively. If the value of the source element is outside the bounds that can be represented in the width of the result element, the result saturates to the minimum or maximum value appropriately. The aggregate result is placed into wrD. Note that participation is not supported for this instruction. Token operation is undefined for this instruction.

Other registers altered:

- None

---

## wpkux - WideWord Pack Unsigned

---

WideWord Unit

**wpkuw**      **wrD, wrA, wrB**

000010	wrD	wrA	wrB	1	00WW	001110
0	5 6	10 11	15 16	20 21 22	26 27	31

Variable values in the following equations are as follows:

WW Value	size	max
01	16	$2^8 - 1$
10	32	$2^{16} - 1$

for  $i = 0$  to  $(128 - (\text{size}/2))$  by  $(\text{size}/2)$

if  $(wrA)_{i \times 2, (i \times 2) + \text{size} - 1} > \text{max}$

$wrD_{i, i + (\text{size}/2) - 1} \leftarrow \text{max}$

else

$wrD_{i, i + (\text{size}/2) - 1} \leftarrow (wrA)_{(i \times 2) + (\text{size}/2), (i \times 2) + \text{size} - 1}$

if  $(wrB)_{i \times 2, (i \times 2) + \text{size} - 1} > \text{max}$

$wrD_{128 + i, 128 + i + (\text{size}/2) - 1} \leftarrow \text{max}$

else

$wrD_{128 + i, 128 + i + (\text{size}/2) - 1} \leftarrow (wrB)_{(i \times 2) + (\text{size}/2), (i \times 2) + \text{size} - 1}$

Let the source vector be the concatenation of the contents of wrA followed by wrB. Each unsigned integer half-word or word, as specified by the WW bits, of the source vector is converted to an unsigned integer byte or half-word, respectively. If the value of the source element is greater than the maximum value that can be represented in the width of the result element, the result saturates to the maximum value. The aggregate result is placed into wrD. Note that participation is not supported for this instruction. Token operation is undefined for this instruction.

Other registers altered:

- None

---

## wprmx - WideWord Permute

---

WideWord Unit

**wprmp**      **wrD, wrA, wrB**

000010	wrD	wrA	wrB	0	PP00	001000
0	5 6	10 11	15 16	20 21 22	25 26	31

for i = 0 to 248 by 8

$$s \leftarrow (wrB)_{i+3, i+7}$$

if PP bits and conditions are set accordingly

$$wrD_{i, i+7} \leftarrow (wrA)_{s \times 8, (s \times 8) + 7}$$

The contents of wrA are the source vector for this permutation operation. Bits 3 to 7 of each byte element of the contents of wrB are used to select a byte element from the source vector for each byte element of the result. The result is placed into wrD, subject to participation. Token operation is undefined for this operation

Other registers altered:

- None

---

## wprmix - WideWord Permute Indirect

WideWord Unit

**wprmix**      **wrD, wrA, rB**

000010	wrD	wrA	rB	0	PP00	001001
0	5 6	10 11	15 16	20 21 22	25 26	31

The following lookup table is used for selecting a permutation vector:

index	vector
0x00	0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
0x01	0x0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00
0x02	0x02030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001
0x03	0x030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102
0x04	0x0405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203
0x05	0x05060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001020304
0x06	0x060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405
0x07	0x0708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203040506
0x08	0x08090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F0001020304050607
0x09	0x090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708
0x0A	0x0A0B0C0D0E0F101112131415161718191A1B1C1D1E1F00010203040506070809
0x0B	0x0B0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A
0x0C	0x0C0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B
0x0D	0x0D0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C
0x0E	0x0E0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D
0x0F	0x0F101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E
0x10	0x101112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F
0x11	0x1112131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10
0x12	0x12131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011
0x13	0x131415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112
0x14	0x1415161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213

index	vector
0x15	0x15161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011121314
0x16	0x161718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415
0x17	0x1718191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213141516
0x18	0x18191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F1011121314151617
0x19	0x191A1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718
0x1A	0x1A1B1C1D1E1F000102030405060708090A0B0C0D0E0F10111213141516171819
0x1B	0x1B1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A
0x1C	0x1C1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B
0x1D	0x1D1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C
0x1E	0x1E1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D
0x1F	0x1F000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E
0x20	0x00020406080A0C0E10121416181A1C1E101030507090B0D0F11131517191B1D1F
0x21	0x010003020504070609080B0A0D0C0F0E111013121514171619181B1A1D1C1F1E
0x22	0x03020100070605040B0A09080F0E0D0C13121110171615141B1A19181F1E1D1C
0x23	0x07060504030201000F0E0D0C0B0A090817161514131211101F1E1D1C1B1A1918
0x24	0x0F0E0D0C0B0A090807060504030201001F1E1D1C1B1A19181716151413121110
0x25	0x1F1E1D1C1B1A191817161514131211100F0E0D0C0B0A09080706050403020100
0x26	0x0002010304060507080A090B0C0E0D0F1012111314161517181A191B1C1E1D1F
0x27	0x0004010502060307080C090D0A0E0B0F1014111512161317181C191D1A1E1B1F
0x28	0x00080109020A030B040C050D060E070F10181119121A131B141C151D161E171F
0x29	0x0001040508090C0D1011141518191C1D020306070A0B0E0F121316171A1B1E1F
0x2A	0x02030001060704050A0B08090E0F0C0D12131011161714151A1B18191E1F1C1D
0x2B	0x06070405020300010E0F0C0D0A0B080916171415121310111E1F1C1D1A1B1819
0x2C	0x0E0F0C0D0A0B080906070405020300011E1F1C1D1A1B18191617141512131011
0x2D	0x1E1F1C1D1A1B181916171415121310110E0F0C0D0A0B08090607040502030001
0x2E	0x000104050203060708090C0D0A0B0E0F101114151213161718191C1D1A1B1E1F
0x2F	0x0001080902030A0B04050C0D06070E0F1011181912131A1B14151C1D16171E1F
0x30	0x0001020308090A0B1011121318191A1B040506070C0D0E0F141516171C1D1E1F
0x31	0x04050607000102030C0D0E0F08090A0B14151617101112131C1D1E1F18191A1B
0x32	0x0C0D0E0F08090A0B04050607000102031C1D1E1F18191A1B1415161710111213
0x33	0x1C1D1E1F18191A1B14151617101112130C0D0E0F08090A0B0405060700010203
0x34	0x0001020308090A0B040506070C0D0E0F1011121318191A1B141516171C1D1E1F
0x35	0x0001020310111213040506071415161708090A0B18191A1B0C0D0E0F1C1D1E1F

---

index	vector
0x36	0x1011121300010203141516170405060718191A1B08090A0B1C1D1E1F0C0D0E0F
0x37	0x08090A0B0C0D0E0F000102030405060718191A1B1C1D1E1F1011121314151617

$index \leftarrow (rB)_{26,31}$

$permvector \leftarrow vector[index]$

for i = 0 to 248 by 8

$s \leftarrow permvector_{i+3, i+7}$

if PP bits and conditions are set accordingly

$wrD_{i, i+7} \leftarrow (wrA)_{s \times 8, (s \times 8) + 7}$

The contents of wrA are the source vector for this permutation operation. The permutation vector is selected from a lookup table using the least significant bits of the contents of rB as an index into the table. Bits 3 to 7 of each byte element of the permutation vector are used to select a byte element from the source vector for each byte element of the result. The result is placed into wrD, subject to participation. Token operation is undefined for this operation

Other registers altered:

- None

---

## wsllx - WideWord Shift Left Logical

---

WideWord Unit

**wsllpw**      **wrD, wrA, wrB**      **(C = 0)**

**wsllcpw**      **wrD, wrA, wrB**      **(C = 1)**

000010	wrD	wrA	wrB	C	PPWW	000000
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

for i = 0 to (256 - size) by size

$$s \leftarrow (wrB)_{i+size-bits, i+size-1}$$

if PP bits and conditions are set accordingly

$$wrD_{i, i+(size-1)} \leftarrow (wrA)_{i+s, i+(size-1)} \parallel 0^s$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted left by the number of bits specified by the low order bits of the corresponding data field contained as contents of wrB, inserting zeros into the low order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ

---

## wsllix - WideWord Shift Left Logical Immediate

WideWord Unit

**wsllipw**      **wrD, wrA, shift\_amount (C = 0)**

**wsllcpw**      **wrD, wrA, shift\_amount (C = 1)**

000010	wrD	wrA	shift_amount	C	PPWW	000010
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

$s \leftarrow \text{shift\_amount}_{5-\text{bits}, 4}$

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i + s, i + (size - 1)} \parallel 0^s$$

The WW field determines if the 256-bit contents of wrA are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted left by the number of bits specified by the appropriate bits of the shift\_amount, inserting zeros into the low order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ



---

## wsrax - WideWord Shift Right Arithmetic

---

WideWord Unit

**wsrapw**      **wrD, wrA, wrB**      **(C = 0)**

**wsracpw**      **wrD, wrA, wrB**      **(C = 1)**

000010	wrD	wrA	wrB	C	PPWW	000101
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

for i = 0 to (256 - size) by size

$$s \leftarrow (wrB)_{i+size-bits, i+size-1}$$

if PP bits and conditions are set accordingly

$$wrD_{i, i+(size-1)} \leftarrow ((wrA)_i)^s \parallel (wrA)_{i, i+size-s-1}$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted right by the number of bits specified by the low order bits of the corresponding data field contained as contents of wrB, sign-extending the high-order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ

---

## wsraix - WideWord Shift Right Arithmetic Immediate

---

WideWord Unit

**wsraipw**     **wrD, wrA, shift\_amount (C = 0)**

**wsraicpw**    **wrD, wrA, shift\_amount (C = 1)**

000010	wrD	wrA	shift_amount	C	PPWW	000111
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

$s \leftarrow \text{shift\_amount}_{5-bits,4}$

for i = 0 to (256 - size) by size

    if PP bits and conditions are set accordingly

$$wrD_{i,i+(size-1)} \leftarrow ((wrA)_i)^s \parallel (wrA)_{i,i+size-s-1}$$

The WW field determines if the 256-bit contents of wrA are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted right by the number of bits specified by the appropriate bits of the shift\_amount, sign-extending the high-order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C =1, WideWord condition code registers: LT, GT, EQ

---

## wsrlx - WideWord Shift Right Logical

---

WideWord Unit

**wsrlpw**      **wrD, wrA, wrB**      **(C = 0)**

**wsrlcpw**      **wrD, wrA, wrB**      **(C = 1)**

000010	wrD	wrA	wrB	C	PPWW	000001
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

for i = 0 to (256 - size) by size

$$s \leftarrow (wrB)_{i+size-bits, i+size-1}$$

if PP bits and conditions are set accordingly

$$wrD_{i, i+(size-1)} \leftarrow 0^s \parallel (wrA)_{i, i+size-s-1}$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted right by the number of bits specified by the low order bits of the corresponding data field contained as contents of wrB, inserting zeros into the high-order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ

---

## wsrlx - WideWord Shift Right Logical Immediate

WideWord Unit

**wsrlpw**      **wrD, wrA, shift\_amount (C = 0)**

**wsrlcpw**    **wrD, wrA, shift\_amount (C = 1)**

000010	wrD	wrA	shift_amount	C	PPWW	000011
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size	bits
00	8	3
01	16	4
10	32	5

$s \leftarrow \text{shift\_amount}_{5-bits, 4}$

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i+(size-1)} \leftarrow 0^s \parallel (wrA)_{i, i+size-s-1}$$

The WW field determines if the 256-bit contents of wrA are treated as 32 bytes, 16 half-words, or 8 words. The contents of each data field of wrA are shifted right by the number of bits specified by the appropriate bits of the shift\_amount, inserting zeros into the high-order bits of each data field of the result. The result is placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C =1, WideWord condition code registers: LT, GT, EQ

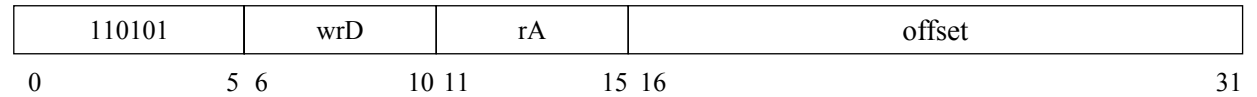
---

## wst - Store WideWord Register

---

WideWord Unit

**wst**                      **wrD, rA, offset**



$$EA \leftarrow 0xFFFFFFFF0 \wedge ((rA) + ((offset_0)^{16} \parallel offset))$$

$$MEM[EA] \leftarrow wrD$$

The 16-bit offset is sign-extended and added to the contents of rA to form the effective address EA. The 256-bit data value and 16-bit token value contents of wrD are stored at the memory location specified by EA (ignoring the least five significant bits to ensure a 256-bit aligned address).

Other registers altered:

- None

---

## wsubx - WideWord Subtract

---

WideWord Unit

**wsubpw**      **wrD, wrA, wrB (C = 0)**

**wsubcpw**    **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	100010
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} + \neg(wrB)_{i, i + (size - 1)} + 1$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate differences of the aligned data fields of wrA and wrB are placed into wrD, subject to participation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ, CA
- A WideWord OV condition code bit is set if the operation in its corresponding datapath causes overflow.

---

## wsubex - WideWord Subtract Extended

---

WideWord Unit

**wsubepw** wrD, wrA, wrB (C = 0)

**wsubecpw** wrD, wrA, wrB (C = 1)

000010	wrD	wrA	wrB	C	PPWW	100011
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} + \neg(wrB)_{i, i + (size - 1)} + CA_{i/8}$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate differences of the aligned data fields of wrA and wrB are placed into wrD, subject to participation. Each data field uses the associated bit of the WideWord Carry register as a carry in for the operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ, CA
- A WideWord OV condition code bit is set if the operation in its corresponding datapath causes overflow.

---

## wsubu - WideWord Subtract Unsigned

WideWord Unit

**wsubupw**    **wrD, wrA, wrB**

000010	wrD	wrA	wrB	1	PPWW	100100
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} + \neg(wrB)_{i, i + (size - 1)} + 1$$

The WW field determines if the 256-bit contents of wrA and wrB are treated as 32 bytes, 16 half-words, or 8 words. The aggregate differences of the aligned data field of wrA and wrB are placed into wrD, subject to participation. This instruction is identical to **wsub** except that the OV condition codes are updated to reflect unsigned arithmetic. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- WideWord condition code registers: LT, GT, EQ, CA
- A WideWord OV condition code bit is set if the operation in its corresponding datapath causes overflow.



---

## wupk<sub>hx</sub> - WideWord Unpack High

---

WideWord Unit

**wupk<sub>hsw</sub>**    **wrD, wrA**    (**C = 0**)

**wupk<sub>huw</sub>**    **wrD, wrA**    (**C = 1**)

000010	wrD	wrA	00000	C	00WW	001101
0	5 6	10 11	15 16	20 21 22	26 27	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - (2 \times \text{size}))$  by  $(2 \times \text{size})$

if  $C=1$

$$wrD_{i, i + (2 \times \text{size}) - 1} \leftarrow 0^{\text{size}} \parallel (wrA)_{i/2, (i/2) + \text{size} - 1}$$

else

$$wrD_{i, i + (2 \times \text{size}) - 1} \leftarrow ((wrA)_{i/2})^{\text{size}} \parallel (wrA)_{i/2, (i/2) + \text{size} - 1}$$

The most significant 128 bits of the contents of wrA are unpacked, or type promoted. For example, if WW=00 the 128-bit source vector is treated as 16 bytes, where each byte is promoted to a 16-bit half-word to form a 256-bit result that is placed into wrD. The C bit indicates whether sign extension or zero fill is used in the unpacking. Note that participation is not supported for this instruction. Token operation is undefined for this instruction

Other registers altered:

- None

---

## wupklx - WideWord Unpack Low

---

WideWord Unit

**wupklsw**    **wrD, wrA**    (**C = 0**)

**wupkluw**    **wrD, wrA**    (**C = 1**)

000010	wrD	wrA	00000	C	00WW	001100
0	5 6	10 11	15 16	20 21 22	26 27	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16

for  $i = 0$  to  $(256 - (2 \times \text{size}))$  by  $(2 \times \text{size})$

if  $C=1$

$$wrD_{i, i + (2 \times \text{size}) - 1} \leftarrow 0^{\text{size}} \parallel (wrA)_{128 + (i/2), 128 + (i/2) + \text{size} - 1}$$

else

$$wrD_{i, i + (2 \times \text{size}) - 1} \leftarrow ((wrA)_{128 + (i/2)})^{\text{size}} \parallel (wrA)_{128 + (i/2), 128 + (i/2) + \text{size} - 1}$$

The least significant 128 bits of the contents of wrA are unpacked, or type promoted. For example, if WW=00 the 128-bit source vector is treated as 16 bytes, where each byte is promoted to a 16-bit half-word to form a 256-bit result that is placed into wrD. The C bit indicates whether sign extension or zero fill is used in the unpacking. Note that participation is not supported for this instruction. Token operation is undefined for this instruction

Other registers altered:

- None

---

## wxorx - WideWord Exclusive-OR

---

WideWord Unit

**wxorpw**      **wrD, wrA, wrB (C = 0)**

**wxorcpw**    **wrD, wrA, wrB (C = 1)**

000010	wrD	wrA	wrB	C	PPWW	101010
0	5 6	10 11	15 16	20 21 22	25 26	31

Variable values in the following equations are as follows:

WW Value	size
00	8
01	16
10	32

for i = 0 to (256 - size) by size

if PP bits and conditions are set accordingly

$$wrD_{i, i + (size - 1)} \leftarrow (wrA)_{i, i + (size - 1)} \oplus (wrB)_{i, i + (size - 1)}$$

The 256-bit contents of wrA are exclusive-ORed with the 256-bit contents of wrB, and the result is placed into wrD, subject to participation. The WW field simply effects how participation applies and how condition codes are updated for this operation. Nominally, the token field of wrA will be written to the token field of wrD. However, some implementations may not ensure this capability.

Other registers altered:

- If C = 1, WideWord condition code registers: LT, GT, EQ

---

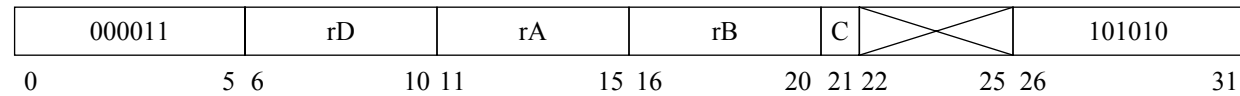
## xorx - Exclusive OR

---

Scalar Unit

**xor**            **rD, rA, rB**    (**C = 0**)

**xorc**            **rD, rA, rB**    (**C = 1**)



$$rD \leftarrow (rA) \oplus (rB)$$

The contents of rA are exclusive-ORed with rB, and the result is placed into rD.

Other registers altered:

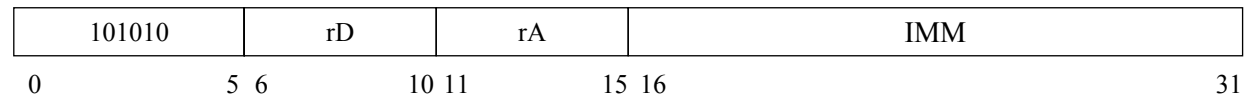
- If C =1, scalar condition code registers: LT, GT, EQ

---

## xori - Exclusive OR Immediate

Scalar Unit

**xori**                      **rD, rA, IMM**



$$rD \leftarrow (rA) \oplus (0^{16} \parallel IMM)$$

The contents of rA are exclusive-ORed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

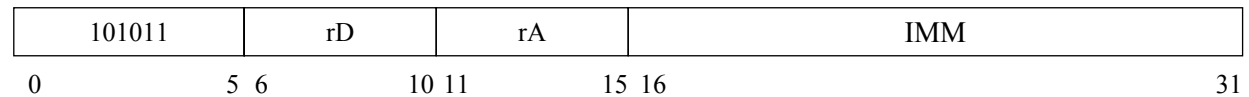
- None

---

## xoric - Exclusive OR Immediate Recording Condition Codes

Scalar Unit

**xoric**            **rD, rA, IMM**



$$rD \leftarrow (rA) \oplus (0^{16} \parallel IMM)$$

The contents of rA are exclusive-ORed with IMM (prepended with zeros to form a 32-bit value), and the result is placed into rD.

Other registers altered:

- Scalar condition code registers: LT, GT, EQ

## **5. Appendix – Phase 2 Digital ASIC Step-Stress and Lifetime Testing Results**

# IRIS Digital Lifetest Reliability Results Summary

Dan Marrujo, Jeff Draper, Jon Osborn  
10-31-2014

© The Aerospace Corporation 2008

## BLUF: Calculated S9 Mean Lifeteime Result

Report Type	ALTA QCP
<b>User Info</b>	
User	Jon Osborn
Company	The Aerospace Corporation
Date	10/8/2014
<b>User Input</b>	
Temperature =	378
Confidence Bounds Used:	2-Sided
Confidence Bounds Method:	Fisher Matrix
Confidence Level =	0.6
<b>ALTA Output</b>	
Upper Bound (0.8) =	5755.896482
Mean Life =	4732.674268 Hr
Lower Bound (0.2) =	3891.349644

Report Type	ALTA QCP
<b>User Info</b>	
User	Jon Osborn
Company	The Aerospace Corporation
Date	10/8/2014
<b>User Input</b>	
Temperature =	378
Confidence Bounds Used:	2-Sided
Confidence Bounds Method:	Fisher Matrix
Confidence Level =	0.9
<b>ALTA Output</b>	
Upper Bound (0.95) =	6939.218643
Mean Life =	4732.674268 Hr
Lower Bound (0.05) =	3227.770571

**S9 Mean Time to Failure (MTTF)**  
**is most likely 4733 Power On Hours**  
**With Vcore= 1.2V, Vio=2.3V and 105C**





## Quantities and Temperatures of IRIS Split Lots Lifetested

Split ID	Temperature (K)			Sub-Totals
	T=398	T=406	T=423	
<b>S9</b>	30	40	30	100
<b>S3</b>	30	40	30	100
<b>POR</b>	20	0	20	40
<b>Sub-Totals:</b>	80	80	80	240

125C                  133C                  150C



## Observations, Inputs, Tools and Analysis

- Most T=0hr fails were removed during packaged part functional screening at TestEdge
- 40 S0, 100 S3 and 100 S9 parts were lifetested at three temperatures
  - S0: 5 of 40 parts failed, 1 failed in first 168hr interval, More fails than expected. Not the focus of this briefing
  - S3: 4 of 100 parts failed, 2 failed in first 168hr interval, most likely T=0+ and random failures.
  - S9: 62 of 100, parts failed, 0 failed in first 168hr interval. Basis of S9 MTTF estimate
- S0, S3 and S9 core logic current and Fmax trends:
  - Core logic lifetested using same temperature/voltage test conditions
  - S0, S3, S9, Core logic all degraded similarly, but did not fail
  - See Verigy Trend Data File: IRIS Life-Test Interactive Chart Hrs-2538.xlsm
- S0, S3 and S9 IO Current Trends:
  - IO voltages were identical for all three test temperatures, Vio=2.3V
  - IO Operating Current (IDD OPER PAD) remained constant ~20mA during test for S0 and S3 (regular DGFET IO Devices)
  - IO Operating Current (IDD OPER PAD) Increased from ~20mA to ~60mA for S9 over the lifetest (Modified DGFET IO Devices)



## Observations, Inputs, Tools and Analysis (cont)

- **S9 Failure Signatures**
  - 3 of 62, S9 parts failed due to internal functional failure. Inability to pass test vectors.
  - 59 of 62, S9 parts failed due to IO continuity failure. Inability to sink or source current through an input or output pin.
- **S9 Voltages, Temperatures, and Times-To-Failure used as data input**
  - *Summary Fail Data File: Time-to-Failure\_10-8-2014.xls*
- **Data Analyzed using Reliasoft Inc. Accelerated Lifetest Analysis (ALTA) Version-9.**
  - *Software lifetest algorithms based on Wayne Nelson, "Accelerated Testing Statistical Models, Test Plans, and Data Analysis", Wiley, 2004.*
  - *Lifetest data best fit by Weibull Distribution.*
  - *Since Vio was not accelerated in this lifetest, de-accelerated to 105C use condition using Arrhenius relation and activation energies from this three temperature life test.*



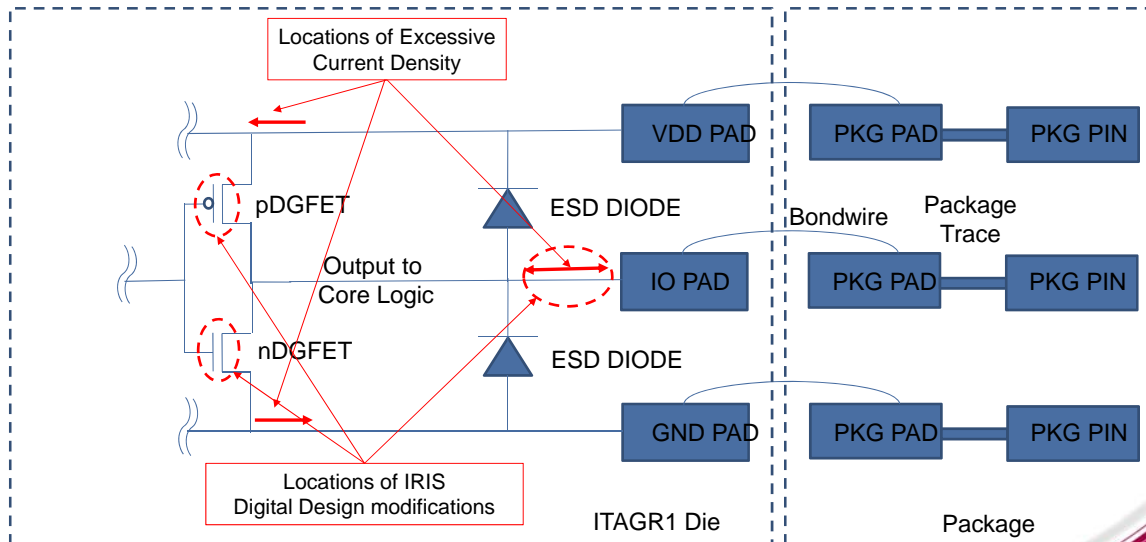
## PoF Inference and Root-Cause Hypothesis

- S0, S3, S9 show similar Fmax Degradation rates under identical stress conditions in Step Stress (SS) and Life Test (LT)
  - *S3 and S9 test results show little affect on core Fmax degradation rate based on SS and LT measurements (consistent with design changes)*
- S0 and S3 show little increase in IO operating current under SS and LT
  - *S0 and S3 test results show little IO reliability sensitivity under LT stress (consistent with design changes)*
- S9 IO supply currents increase rapidly under constant voltage stress during SS testing and during LT
  - *As a result of S9 testing, initially IO current increases rapidly, consistent with GO-TDDB degradation, leading to high internal gate leakage currents (consistent with design changes)*
  - *Subsequently after a latency period S9 continuity failures occur*
  - *Continuity failures are consistent with Electromigration (EM) of IO interconnect to IO PAD (consistent design changes)*

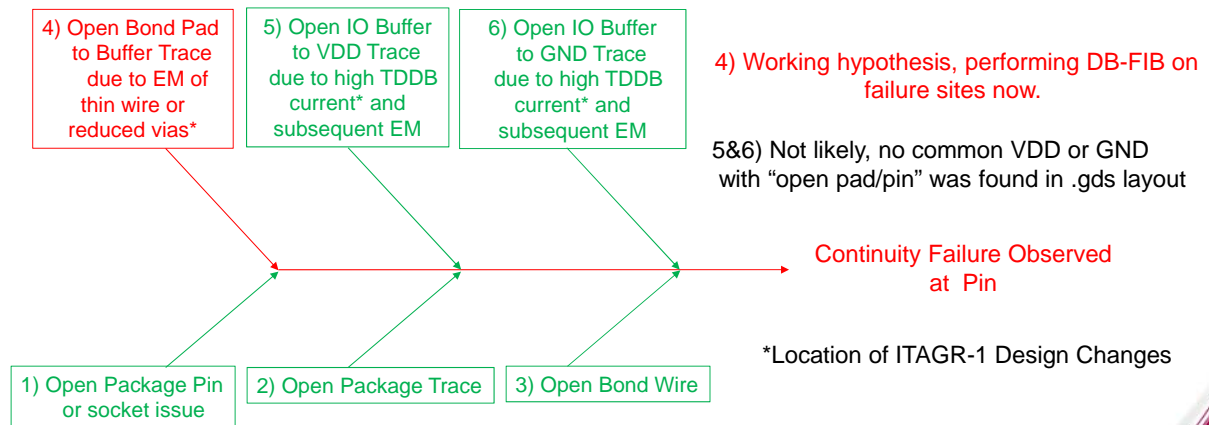
**S9 SS and LT results are consistent with a two failure mechanisms: GO-TDDB and EM degradation processes, leading to device failures observed in DARPA IRIS lifetest**



## Simplified ITAGR-1 Output PAD Circuit Diagram



## Fish-Bone Diagram of S9 Continuity Failures

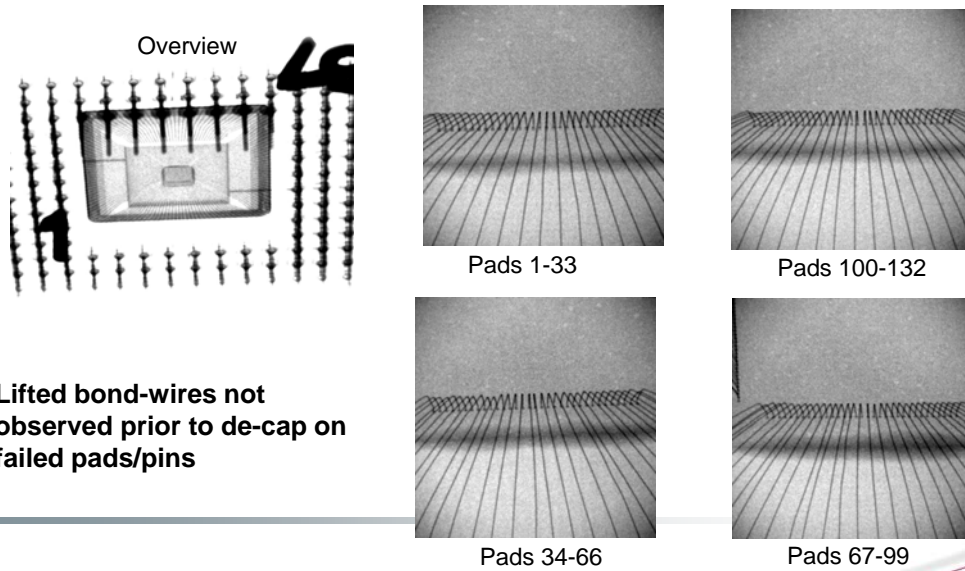


1) Not likely as part was removed, pin inspected and inserted multiple times with same fail signature.

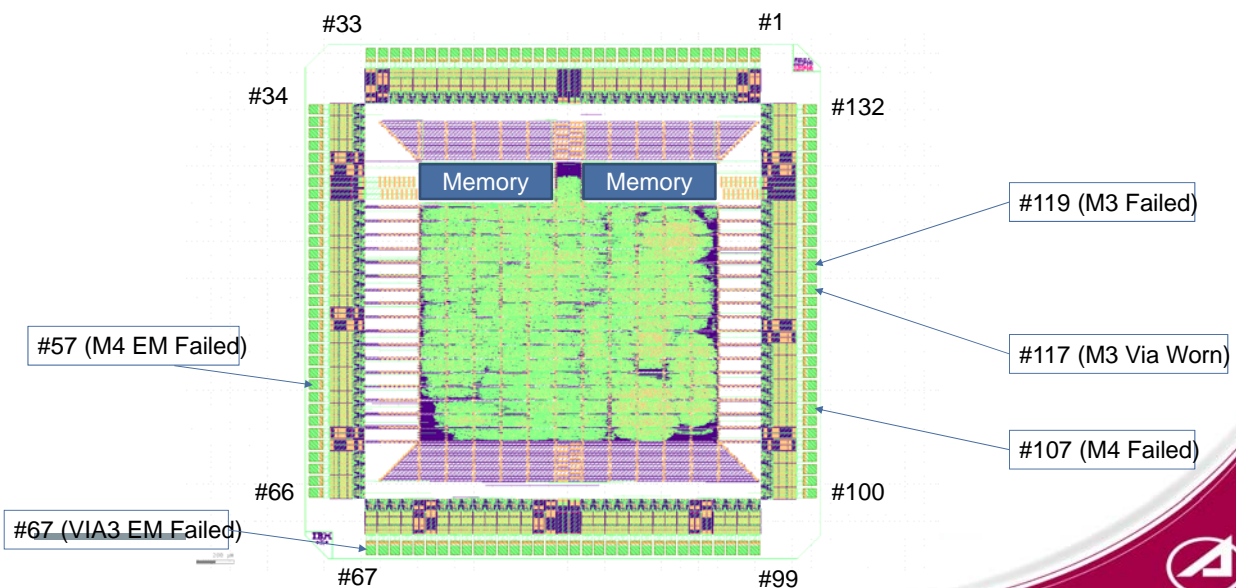
2) Not likely as part was x-ray inspected no open trace was observed in package traces.

3) Not likely as part was x-ray inspected no open/lifted bondwire was observed in failed parts

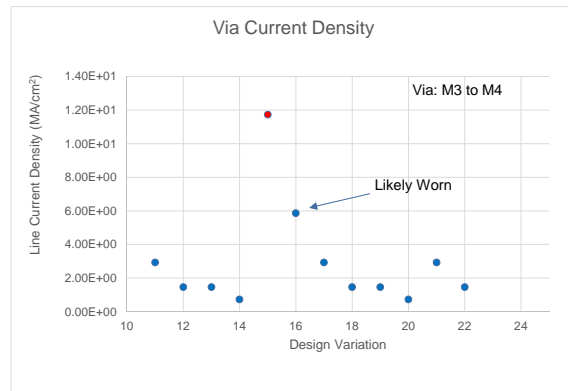
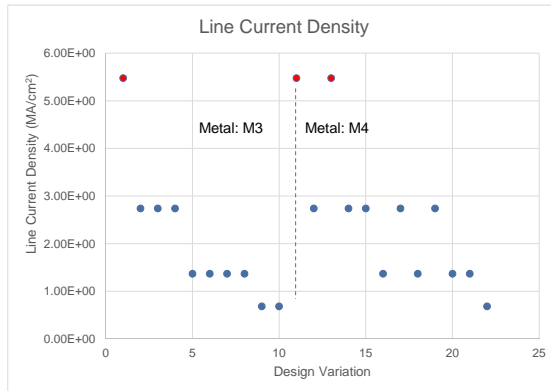
## Aerospace Bond-Wire x-Ray Inspection Images (one of four parts, typical result, supporting conclusion #3 of fish-bone)



## Overall Layout of 90nm ITAGR-1 Processor (EM Sites)



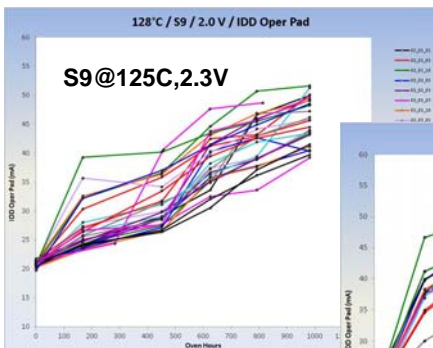
## Est. Current Density In ITAGR-1 Thin Output Wires and Reduced Via Design Variants



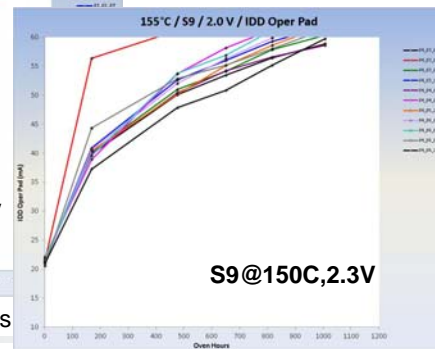
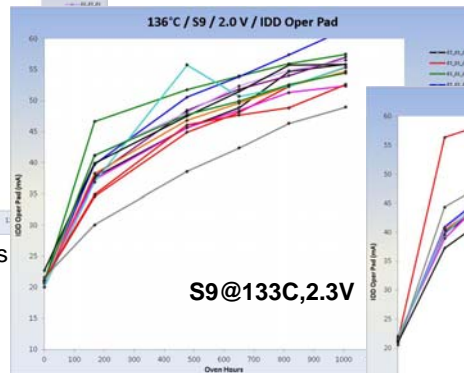
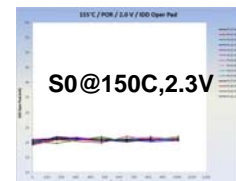
**Note: Red "Dots" are associated with the most common failure pads (#57, #107, #119, #67)**



## S9 Verigy Data: IO Degradation, Current vs. Time

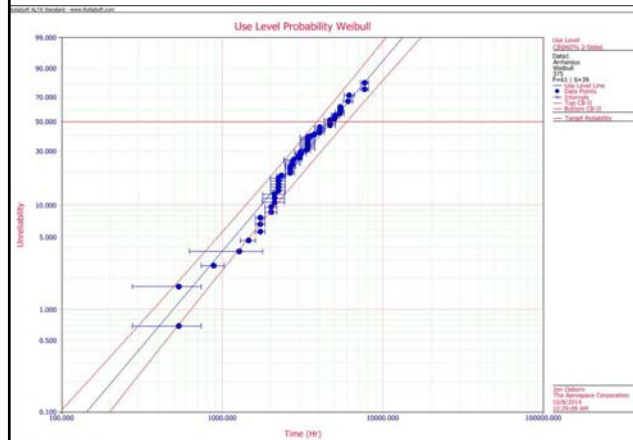


- IO is under constant voltage stress
- Vio-min=2.3V, Three Temperatures
- Inputs Driven, Outputs are Unloaded
- Test Condition for Best Case IO Lifetime

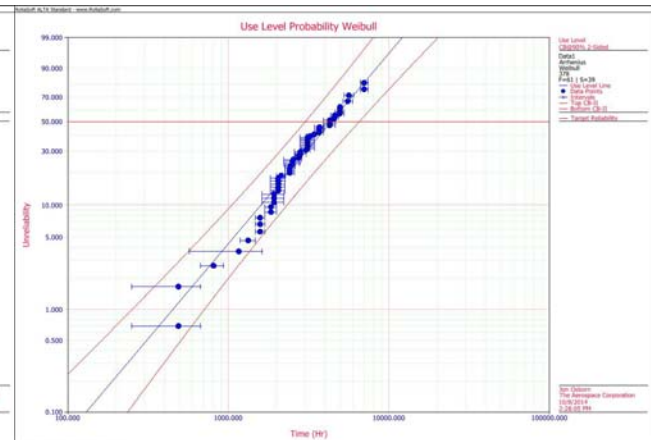


## S9 Weibull plots, max. use condition: 105C

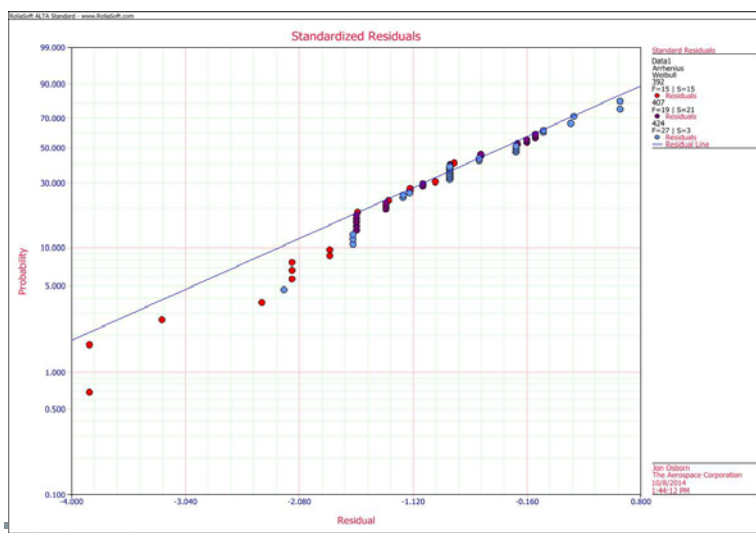
60% Confidence Bounds Shown



90% Confidence Bounds Shown



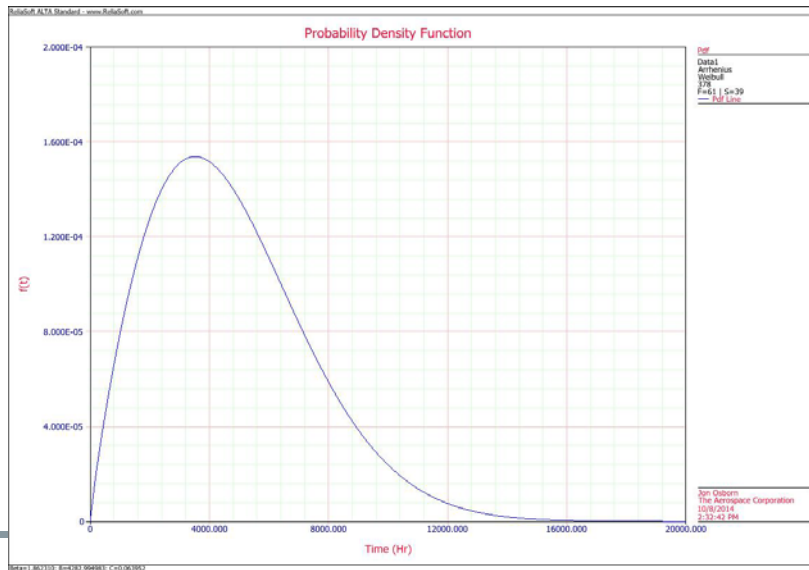
## S9 Weibull Standardized Residuals



- Weibull distribution Fits lifetest data well
- Alternative Log-Normal fit was also investigated, but Weibull provided smaller error residuals

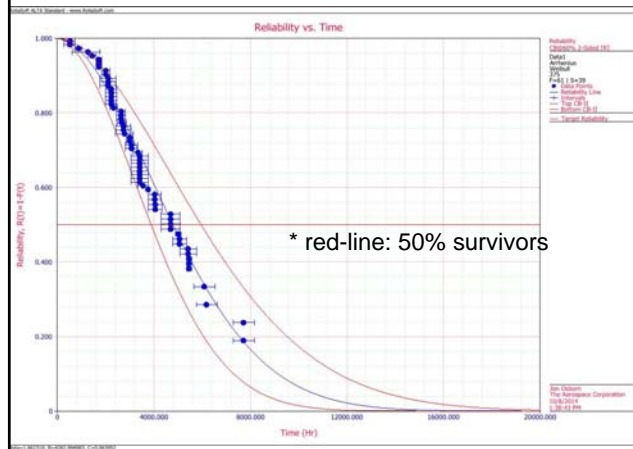


## S9 Weibull Probability Density Function

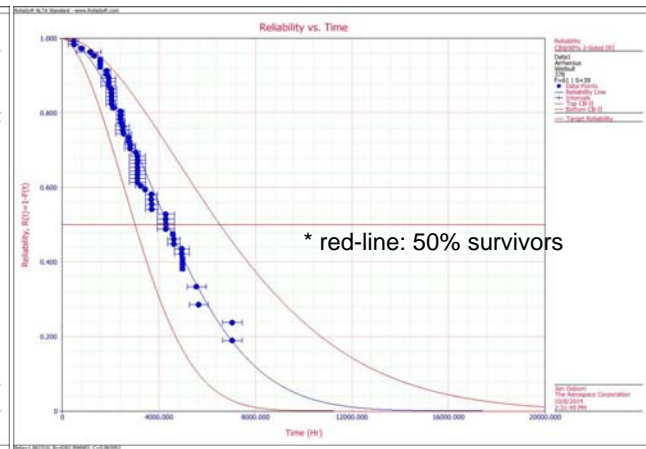


## S9 Reliability vs Time Plots at 105C

60% Confidence Bounds Shown

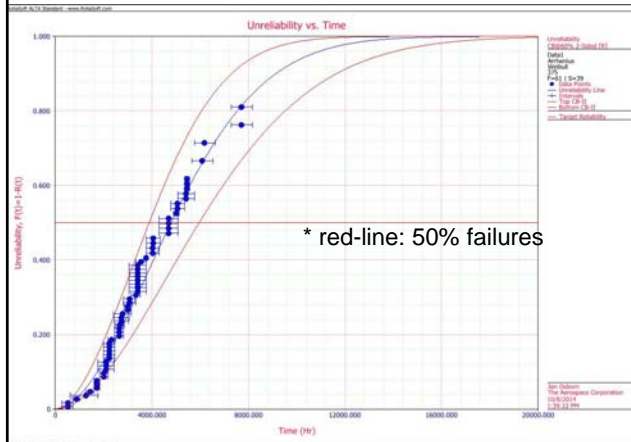


90% Confidence Bounds Shown

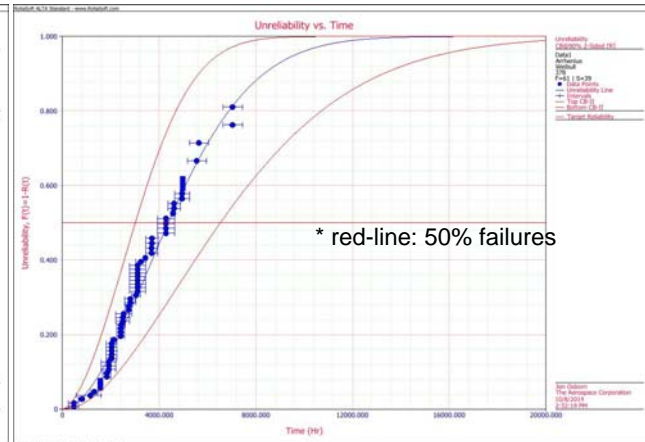


## S9 Unreliability vs Time Plots at 105C

60% Confidence Bounds Shown



90% Confidence Bounds Shown



## Calculated S9 Mean Lifetime Result

Report Type	ALTA OCP
<b>User Info</b>	
User	Jon Osborn
Company	The Aerospace Corporation
Date	10/8/2014
<b>User Input</b>	
Temperature =	378
Confidence Bounds Used:	2-Sided
Confidence Bounds Method:	Fisher Matrix
Confidence Level =	0.6
<b>ALTA Output</b>	
Upper Bound (0.8) =	5755.896482
Mean Life =	4732.674268 Hr
Lower Bound (0.2) =	3891.349644

Report Type	ALTA OCP
<b>User Info</b>	
User	Jon Osborn
Company	The Aerospace Corporation
Date	10/8/2014
<b>User Input</b>	
Temperature =	378
Confidence Bounds Used:	2-Sided
Confidence Bounds Method:	Fisher Matrix
Confidence Level =	0.9
<b>ALTA Output</b>	
Upper Bound (0.95) =	6939.218643
Mean Life =	4732.674268 Hr
Lower Bound (0.05) =	3227.770571

S9 Mean Time to Failure (MTTF)  
is most likely 4733 Power On Hours  
with Vcore= 1.2V, Vio=2.3V and 105C



## Backup Charts



## Aerospace DB-FIB EM-site Cross-Section Images

Insert DB-FIB EM-site Images here



## Aerospace HR-TEM DGFET Gate-Oxide Cross-Section Images

Insert HRTEM DGFET Gate-Oxide here



## Aerospace HR-TEM EM-site Plan View Images

Insert HRTEM EM-site Images here

